The Islamic University of Gaza
Electrical Engineering Department
2013-2014 Fall semester

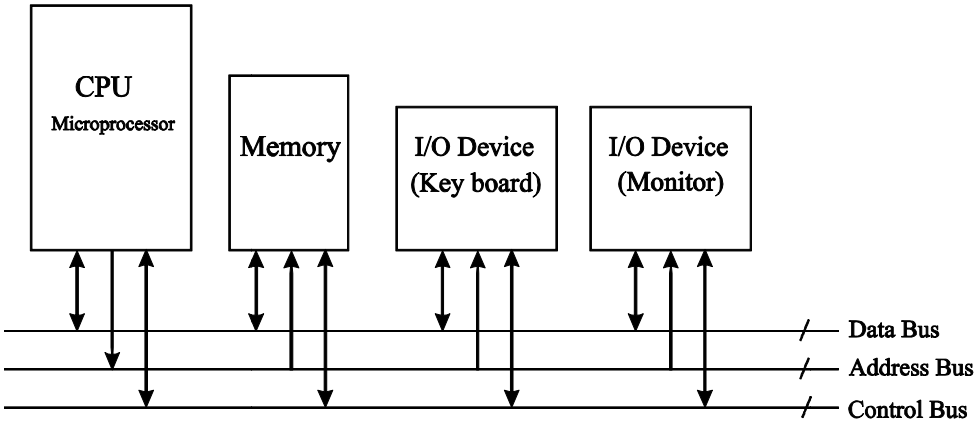
Course title & Code	Microcontroller Systems Design (EELE 4315)	
Instructor	Prof. Dr. Muhammed ABDELATI Room B230 Tel 2878 E-mail muhammet@iugaza.edu.ps	
Teaching Assistant	Eng. Ahmed Afifi E-mail ahafifi@iugaza.edu	
Prerequisites	This course is dedicated to 4th year electrical engineering students who completed successfully the Computer Programming and Logic Design courses.	
Office Hours	Saturday 08:30-11:00 (♀) Monday 08:30-11:00 (♂) These office hours are tentative. Official ones are used to be announced on my office door.	
Course Objectives	The course provides a thorough description of the design and implementation of microcontrollers and microprocessors systems.	
Text Books	J. Sanchez and M. Canton , "Microcontroller Programming – The Microchip PIC", 2007 Nebojsa Matic, "The PIC Microcontrollers ", 2000 Dogan Ibrahim, "Advanced PIC Microcontroller Projects in C" , 2008	
Recommended References	PIC16F84A Datasheet MicroC User's Manual	
Grading (Tentative)	Midterm Examination on Sunday 27/10/2013 at class time Attendance & Participation Final Examination	40 20 40
Exams Dates	The above dates are fixed. It is your own responsibility to arrange other exams and labs accordingly. For unsolvable situations, you have to consult me before 22/09/2013.	
Grading Policy	• Cheating and late submissions are severely penalized. You will lose at least **25% per day** for late submissions of your homework and projects. • The Attendance and Participation grade is calculated according to the "Second Chance System (SCS)" which will be used during the course. Basically, your record is updated during each class in three lists: the Attendance List, the White List (positive behavior), and Black List (negative behavior such as coming late, having your mobile phone ringing, ….). Details of the SCS will be given at class but for now you should know that missing each class hour costs you 1 point and having two records in the Black List costs you 1 point. • In exams a special bonus agreement may be adopted. • Missing more 5 hour meetings without official excuse is your easiest way to fail in the course.	

Course Outline

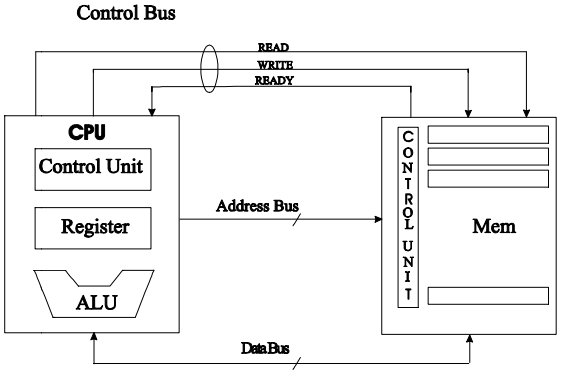
Week	Subject
1-2	Statement of course objectives, contents, methods, and evaluation Basic Computer Architectures Instruction execution and pipelining Microcontrollers versus microprocessors Harvard versus Von-Neumann's Architecture (RISC, CISC) PIC16F84A Microcontroller Implementing first Application Software & Hardware Development Tools
3-4	PIC Microcontrollers Families Source Code Format Addressing modes Delay Loops Interrupts
5-6	Lockup tables EEPROM utilization Macros
7-8	Midterm Programming PIC microcontrollers in C Flasher Application in C (Simple) Iqama Application in C (Using interrupts)
9-10	Keypad and LCD Interface Analog interface
11-12	Serial Communication DC motor control Stepper Motor Control
13-15	Selected Projects

Chapter 1 - Microcontroller Basics

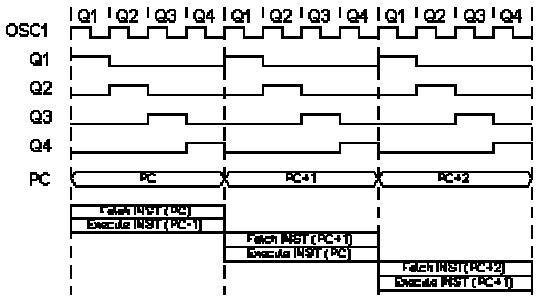
1.1 Simple Computer Architecture



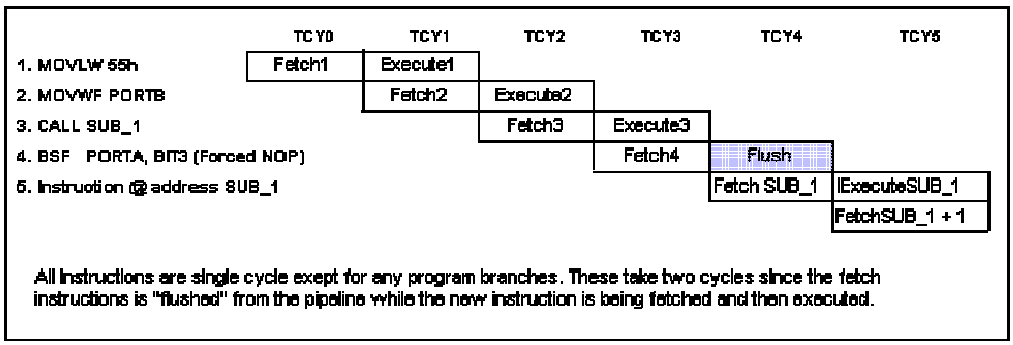
1.2 Instruction Handling Sequence (Fetch-Execute)



1.3 Pipelining



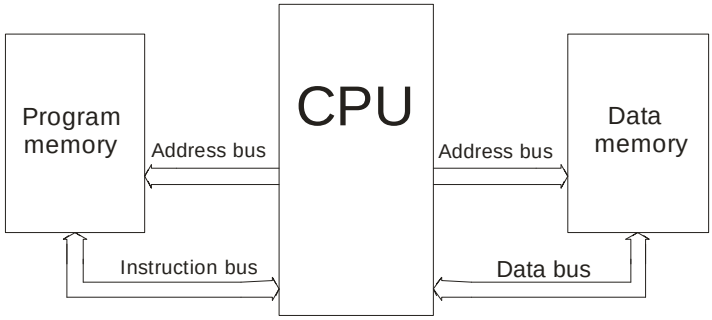
Clock/Insruction Cycle



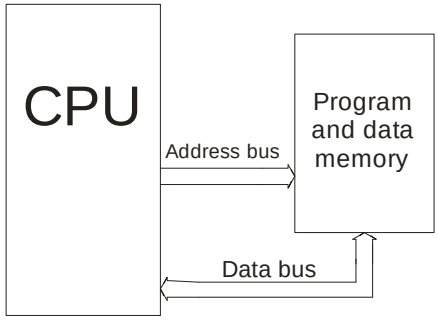
Instruction Pipeline Flow

1.4 Microcontrollers versus microprocessors

1.5 Harvard versus Von-Neumann's Architecture (RISC, CISC)

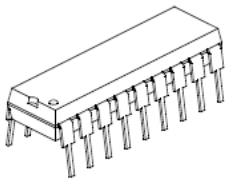


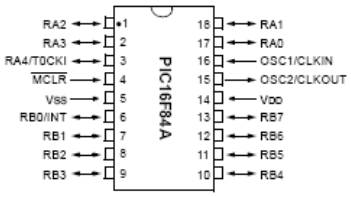
Harvard (RISC)



Von-Neuman CISC

1.6 PIC 16F84A





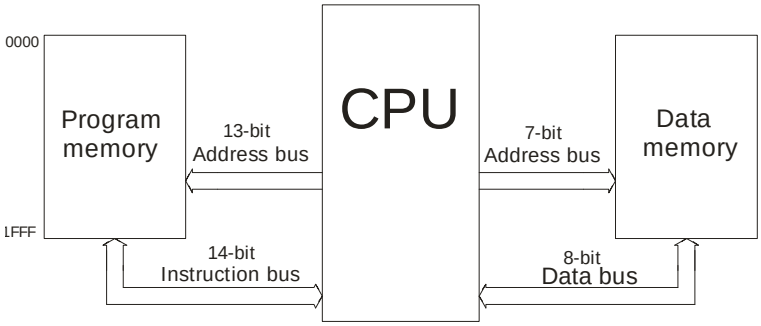
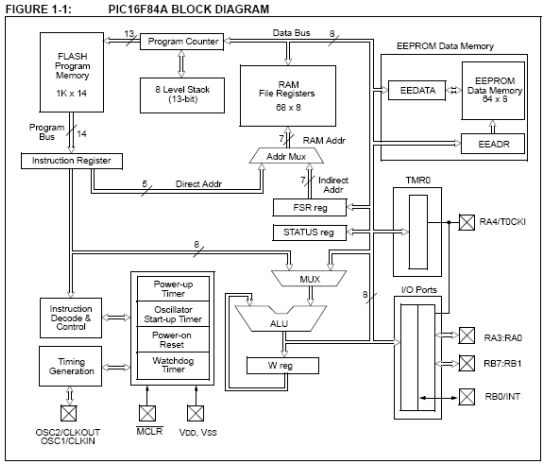


FIGURE 1-1: PIC16F84A BLOCK DIAGRAM



1.6.1 Data memory

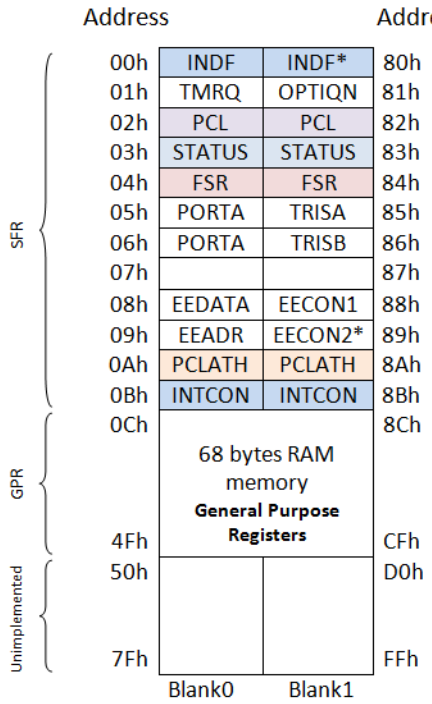


TABLE 2-1: SPECIAL FUNCTION REGISTER FILE SUMMARY

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on RESET	Details on page
Bank 0											
00h	INDF	Uses contents of FSR to address Data Memory (not a physical register)								---- ----	11
01h	TMR0	8-bit Real-Time Clock/Counter								xxxx xxxx	20
02h	PCL	Low Order 8 bits of the Program Counter (PC)								0000 0000	11
03h	STATUS(2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	8
04h	FSR	Indirect Data Memory Address Pointer 0								xxxx xxxx	11
05h	PORTA(4)	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	16
06h	PORTB(5)	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	18
07h	—	Unimplemented location, read as '0'								—	—
08h	EEDATA	EEPROM Data Register								xxxx xxxx	13,14
09h	EEADR	EEPROM Address Register								xxxx xxxx	13,14
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of the PC(1)					---0 0000	11
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10
Bank 1											
80h	INDF	Uses Contents of FSR to address Data Memory (not a physical register)								---- ----	11
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	9
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	11
83h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	8
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	11
85h	TRISA	—	—	—	PORTA Data Direction Register					---1 1111	16
86h	TRISB	PORTB Data Direction Register								1111 1111	18
87h	—	Unimplemented location, read as '0'								—	—
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	13
89h	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	14
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC(1)					---0 0000	11
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0', q = value depends on condition

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> are never transferred to PCLATH.

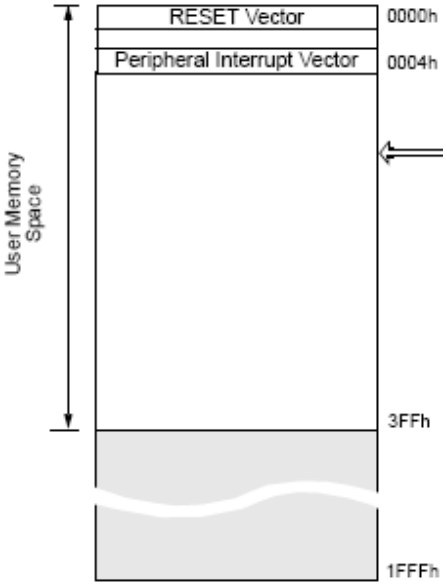
2: The $\overline{TO}$ and $\overline{PD}$ status bits in the STATUS register are not affected by a $\overline{MCLR}$ Reset.

3: Other (non power-up) RESETS include: external RESET through $\overline{MCLR}$ and the Watchdog Timer Reset.

4: On any device RESET, these pins are configured as inputs.

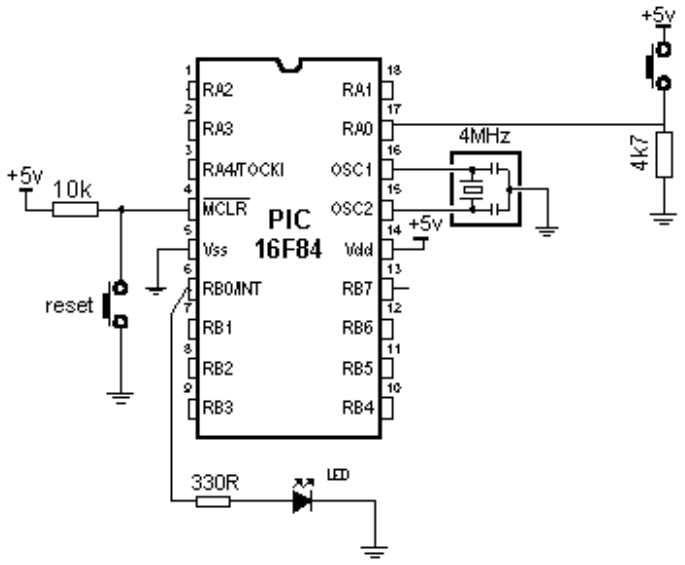
5: This is the value that will be in the port output latch.

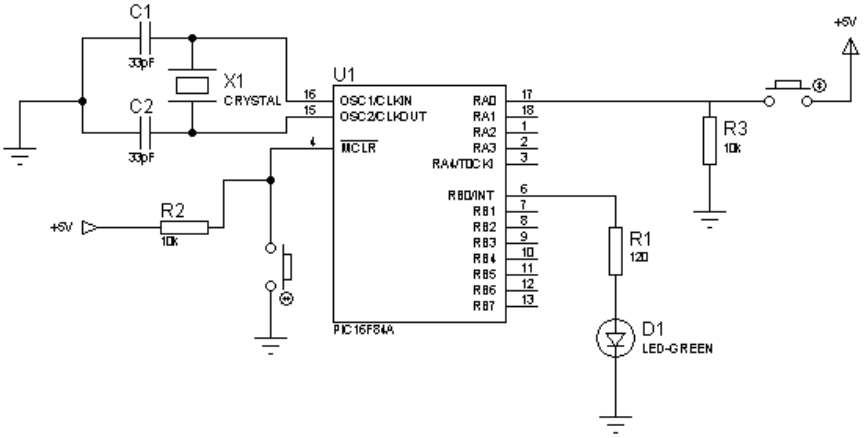
1.6.2 Program Memory



1.7 Implementing Your First Application

1.7.1 Hardware connection



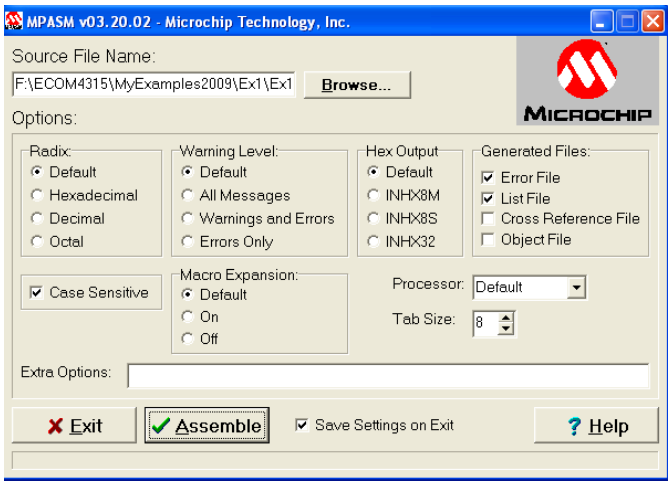


1.7.2 Software program

```
;B0=A0
                LIST   P=PIC16F84A
                #INCLUDE <P16F84A.INC>
                __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

                ORG 0X000
                GOTO START

                ORG 0X004
                RETFIE

START           BSF   STATUS,RP0        ;BANK1
                BCF   TRISB,0          ;RB0 IS OUTPUT
                BSF   TRISA,0          ;RA0 IS INPUT
                BCF   STATUS,RP0        ;BANK0

LOOP            MOVF  PORTA,W
                MOVWF PORTB
                GOTO  LOOP

                END
```

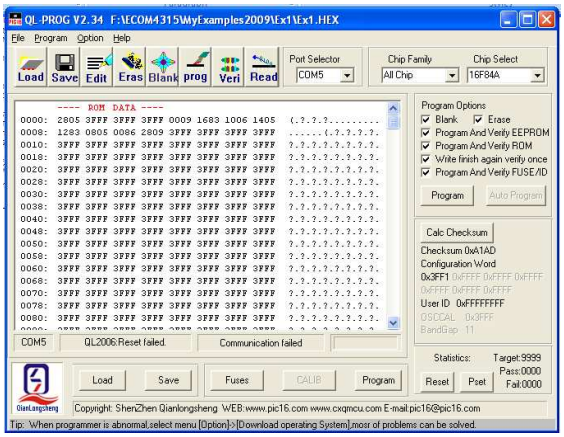
1.7.3 Program compilation Tools

MPLAB, MPASM



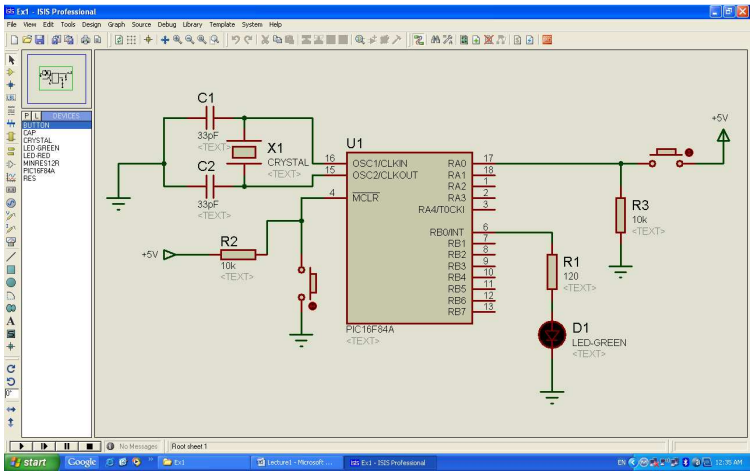
1.7.4 Program Loading Tools

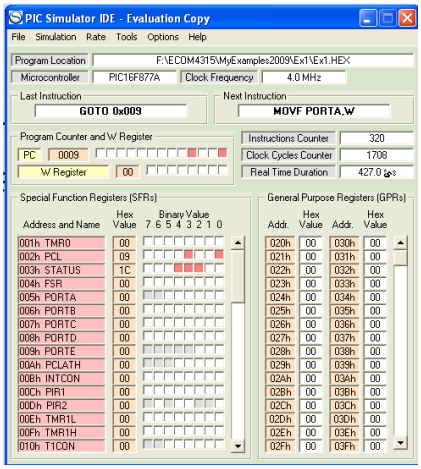
WINPIC800, QL-PROG



1.7.5 Simulation tools

Proteus, PIC Simulator IDE





Chapter 2 - Programming microcontrollers using Assembly language

2.1 Machine language and Assembly language:

Computer: is a machine that can solve problems by carrying instructions given to it.
Program: is a sequence of instructions describing how to perform a certain task.
Machine language: is the computer primitive instructions consisting of 0's and 1's.

This program copies A0 to B0

0000000000000	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0000	28	05
0000000000000	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0001	xx	xx
0000000000001	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0002	xx	xx
0000000000001	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0003	xx	xx
0000000000010	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0004	00	09
0000000000010	0	1	0	1	1	0	1	0	0	0	0	0	1	1	0005	16	83
0000000000011	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0006	10	06
0000000000011	0	1	0	1	0	0	0	0	0	0	0	1	0	1	0007	14	05
0000000000100	0	1	0	0	1	0	1	0	0	0	0	0	1	1	0008	12	83
0000000000100	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0009	08	05
0000000000101	0	0	0	0	0	0	1	0	0	0	0	1	1	0	000A	00	86
0000000000101	1	0	1	0	0	0	0	0	0	0	1	0	1	1	000B	28	0
0000000000110	x	x	x	x	x	x	x	x	x	x	x	x	x	x	000C	x	x
0000000000110	x	x	x	x	x	x	x	x	x	x	x	x	x	x	000D	x	x
…															…		
1111111111111															1FFF		
1000000000011	1	1	1	1	1	1	1	1	1	1	0	0	0	1	2007	3F	F1

It is almost impossible to write programs directly in machine language. However, they may be written in a human readable form of the machine language which is called the ***Assembly language***

```
        GOTO   START

        RETFI
START   BSF    0x03,0x5
        BCF    0x86,0x0
        BSF    0x85,0x0
        BCF    0x03,0x5
LOOP    MOVF   0x05,0
        MOVWF  0x06
        GOTO   LOOP
```

```
GOTO   START
RETFI
BSF    STATUS,RP0
BCF    TRISB,0
BSF    TRISA,0
BCF    STATUS,RP0
MOVF   PORTA,W
MOVWF  PORTB
GOTO   LOOP
```

0000	28	05
0001	xx	xx
0002	xx	xx
0003	xx	xx
0004	00	09
0005	16	83
0006	10	06
0007	14	05
0008	12	83
0009	08	05
000A	00	86
000B	28	0B

The* P16F84a.INC *file allowed us to write identifiers for memory locations instead of their addresses

2.2 Assembler Statements (Instructions / Directives) and numbers Representation

Instructions are translated to machine code while ***Directives*** directs the assembler during translation. For example the END directive informs the assembler to stop.

In assembly language, numbers can be represented in decimal, hexadecimal or binary form. We will illustrate this with a number 240:

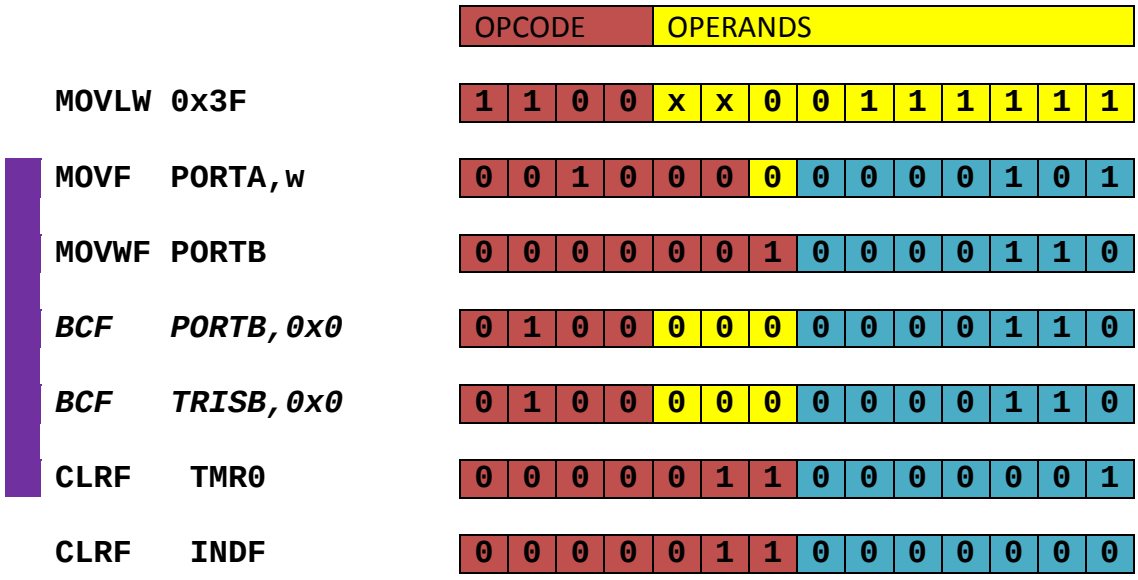
.240	decimal
0xF0	hexadecimal
b'11110000'	binary

2.3 Addressing Modes

The PIC 16F84A µcontroller has 35 machine language instructions each one is referred by 3-6 letters mnemonics in the assembly language. The mnemonics may be followed by one or more operands.

The machine code of each instruction is divided into fields:

- ⊗ ***Operation code (op code):*** indicating what the processor is to do.
- ⊗ ***Operands:*** indicating the information needed by the instruction (datum or its location)



The way in which an operand is specified is called its addressing mode. There are 2 basic addressing modes:

1. Immediate operand **XORLW 0x5F**

2. Memory operand

Direct **CLRF TMR0**

Indirect **CLRF INDF**

2.3.1 Expanding the address space through the idea of Banks while using Direct & Indirect Addressing

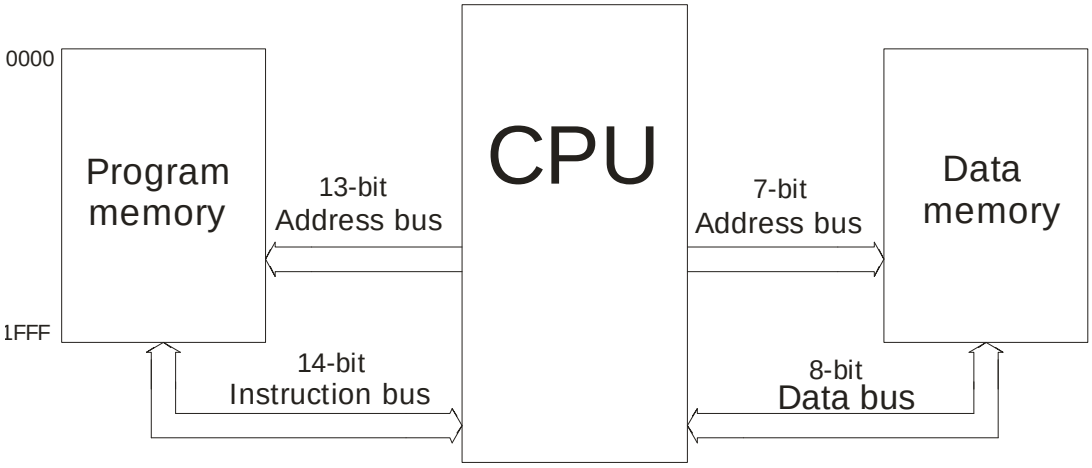
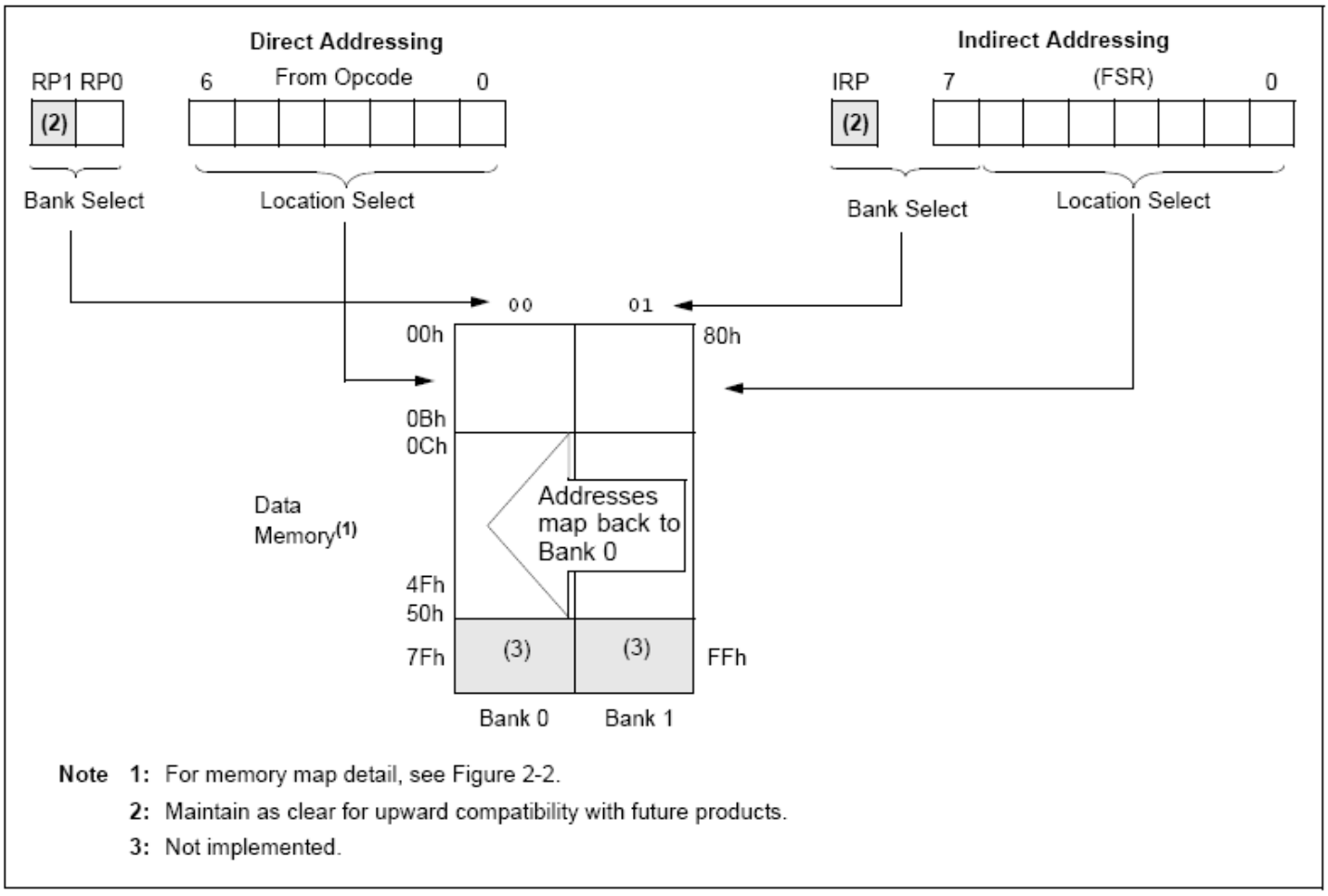


FIGURE 2-3: DIRECT/INDIRECT ADDRESSING



Note 1: For memory map detail, see Figure 2-2.
2: Maintain as clear for upward compatibility with future products.
3: Not implemented.

2.3.2 Status register

REGISTER 2-1: STATUS REGISTER (ADDRESS 03h, 83h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

bit 7-6 **Unimplemented:** Maintain as '0'

bit 5 **RP0**: Register Bank Select bits (used for direct addressing)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)

bit 4 $\overline{TO}$: Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 3 $\overline{PD}$: Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2 Z: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

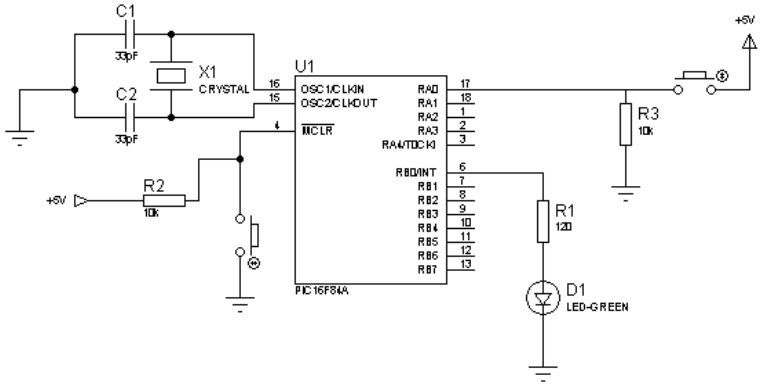
bit 1 DC: Digit carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions) (for $\overline{borrow}$, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

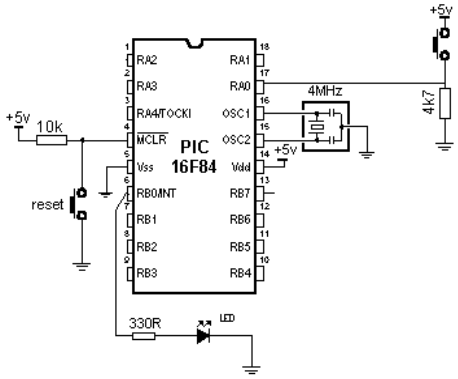
bit 0 C: Carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions) (for $\overline{borrow}$, the polarity is reversed)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high or low order bit of the source register.

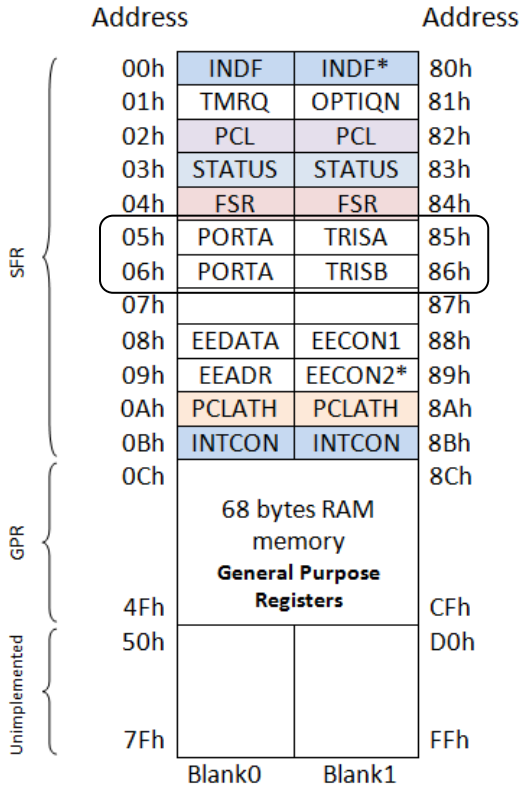
Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

2.3.3 PORTA, TRISA, PORTB, TRISB registers



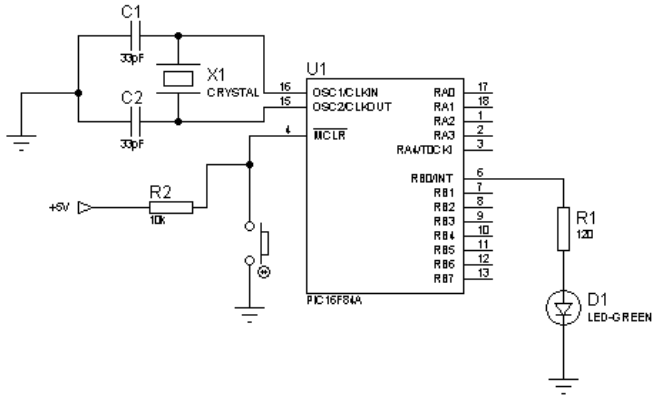



```
;Ex1 (Buffer) B0=A0
                LIST   P=PIC16F84A
                #INCLUDE <P16F84A.INC>
                __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

                ORG 0X000
                GOTO START

                ORG 0X004
                RETFIE

START           BSF   STATUS,RP0        ;BANK1
                BCF   TRISB,0           ;RB0 IS OUTPUT
                BSF   TRISA,0           ;RA0 IS INPUT
                BCF   STATUS,RP0        ;BANK0

LOOP            MOVF  PORTA,W
                MOVWF PORTB
                GOTO  LOOP

                END
```



2.4 Using loops

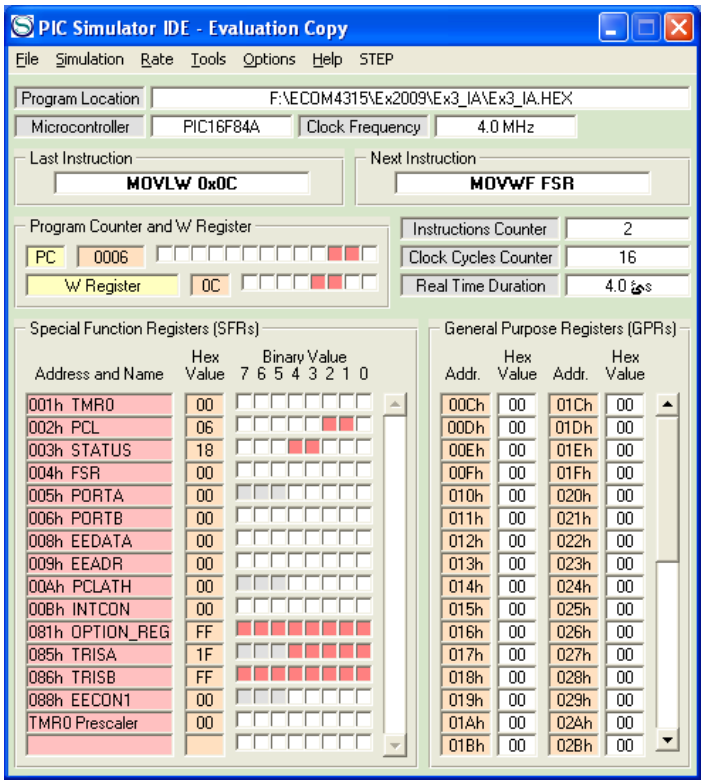
2.4.1 to generate delays



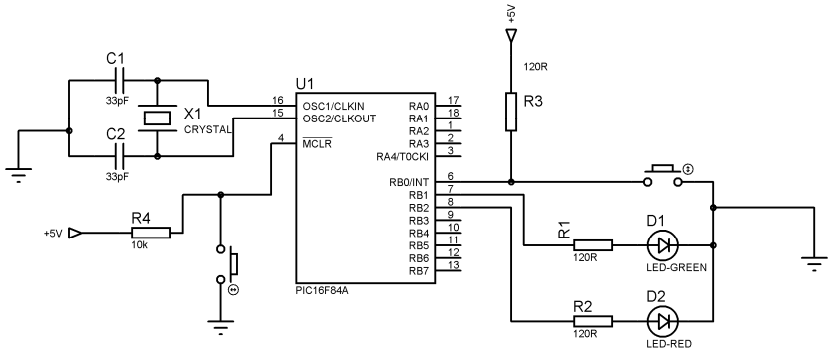

```
;Ex2 (Flasher) on B0
           LIST   P=PIC16F84A
           #INCLUDE <P16F84A.INC>
           __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

           ORG    0X000
           GOTO   START

           ORG    0X004
           RETFIE

START      BSF    STATUS,RP0          ;BANK1
           BCF    TRISB,0             ;RB0 IS OUTPUT
           BCF    STATUS,RP0          ;BANK0

           MOVLW  01h
LOOP       XORWF  PORTB, F
           CALL   DELAY
           GOTO   LOOP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
DELAY      MOVLW  .255
           MOVWF  0CH
LOOP1      NOP
           MOVLW  .255
           MOVWF  0DH
LOOP0      NOP
           DECFSZ 0DH
           GOTO   LOOP0

           DECFSZ 0CH
           GOTO   LOOP1
           RETURN
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
           END
```

2.4.2 to process memory arrays

```
;Ex3 (Memory Block Processing)
;Indirect Addressing application
;Filling All GPR registers with 00h

            LIST   P=PIC16F84A
            #INCLUDE <P16F84A.INC>
            __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

            ORG    0X000
            GOTO   START
            ORG    0X004
            RETFIE

START       MOVLW 0Ch
            MOVWF FSR
LOOP1       CLRF  INDF
            INCF  FSR
            MOVLW 50h
            SUBWF FSR,w
            BTFSS STATUS,Z
            GOTO  LOOP1

LOOP2       GOTO  LOOP2
            END
```



2.5 Interrupts




```
;Ex4 (Interrupt application)
           LIST   P=PIC16F84A
           #INCLUDE <P16F84A.INC>
           __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

           ORG    0X000
           GOTO   START

           ORG    0X004
           MOVWF  0Eh             ;push W and Status
           MOVF   STATUS,W
           MOVWF  0Fh

           BCF    STATUS,RP0
           MOVLW  4
           XORWF  PORTB,F
           BCF    INTCON,INTF     ;Reset the Interrupt Flag

           MOVF   0Fh,W           ;pop W and Status
           MOVWF  STATUS
           SWAPF  0EH,F
           SWAPF  0EH,W
           RETFIE

START      BSF    STATUS,RP0        ; BANK 1
           BCF    OPTION_REG,6      ; -0-- ----  FOR Edge programmig
           BSF    TRISB,0           ; B0 IS INPUT

           MOVLW  B'10010000'
           MOVWF  INTCON            ; 1--1 --0-  CLEAR EXTERNAL INT FLAG

           BCF    TRISB,1           ; B1 is output
           BCF    TRISB,2           ; B2 is output
           BCF    STATUS,RP0        ; BANK0

LOOP       MOVLW  02h
           XORWF  PORTB,F
           CALL   DELAY
           GOTO   LOOP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
DELAY      MOVLW  .255
           MOVWF  0CH
LOOP1      NOP
           MOVLW  .255
           MOVWF  0DH
```

Interrupts are a mechanism of a microcontroller which enables it to respond to some events at the moment they occur, regardless of what microcontroller is doing at the time.

Each interrupt changes the program flow, interrupts it and after executing an interrupt subroutine it continues from that same point on.

2.5.1 Sources of interrupt for PIC16F84A:

1. Termination of writing data to EEPROM.
2. TMR0 interrupt caused by timer overflow.
3. Interrupt during alteration on RB4, RB5, RB6 and RB7 pins of port B.
4. External interrupt from RB0/INT pin of microcontroller.

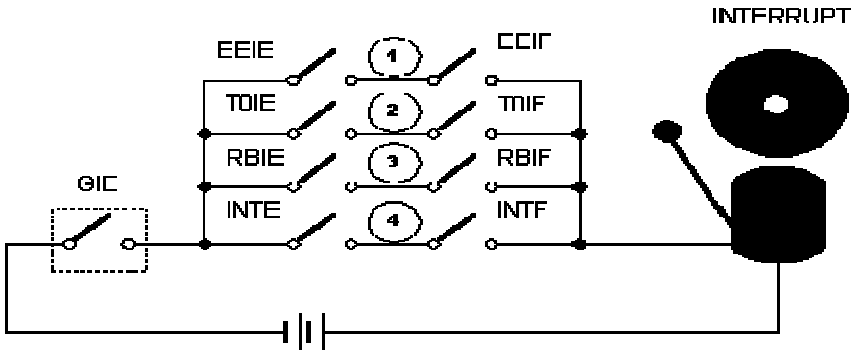
2.5.2 Registers used with interrupt:

1. Control interrupt register (INTCON) at 0Bh address.
2. Option register at 81h address.
3. Control EEPROM register (EECON 1) at 88h address.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

bit 7	**GIE:** Global Interrupt Enable bit 1 = Enables all unmasked interrupts 0 = Disables all interrupts
bit 6	**EEIE**: EE Write Complete Interrupt Enable bit 1 = Enables the EE Write Complete interrupts 0 = Disables the EE Write Complete interrupt
bit 5	**T0IE**: TMR0 Overflow Interrupt Enable bit 1 = Enables the TMR0 interrupt 0 = Disables the TMR0 interrupt
bit 4	**INTE**: RB0/INT External Interrupt Enable bit 1 = Enables the RB0/INT external interrupt 0 = Disables the RB0/INT external interrupt
bit 3	**RBIE**: RB Port Change Interrupt Enable bit 1 = Enables the RB port change interrupt 0 = Disables the RB port change interrupt
bit 2	**T0IF**: TMR0 Overflow Interrupt Flag bit 1 = TMR0 register has overflowed (must be cleared in software) 0 = TMR0 register did not overflow
bit1	**INTF**: RB0/INT External Interrupt Flag bit 1 = The RB0/INT external interrupt occurred (must be cleared in software) 0 = The RB0/INT external interrupt did not occur
bit 0	**RBIF**: RB Port Change Interrupt Flag bit 1 = At least one of the RB7:RB4 pins changed state (must be cleared in software) 0 = None of the RB7:RB4 pins have changed state

Option Register	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS1
EECON1 Register				EEIF	WRERR	WREN	WR	RD



When an interrupt occurs, its corresponding flag bit is set. It must be cleared in software in order to allow repeating interrupt events.

2.5.3 Keeping the contents of important registers

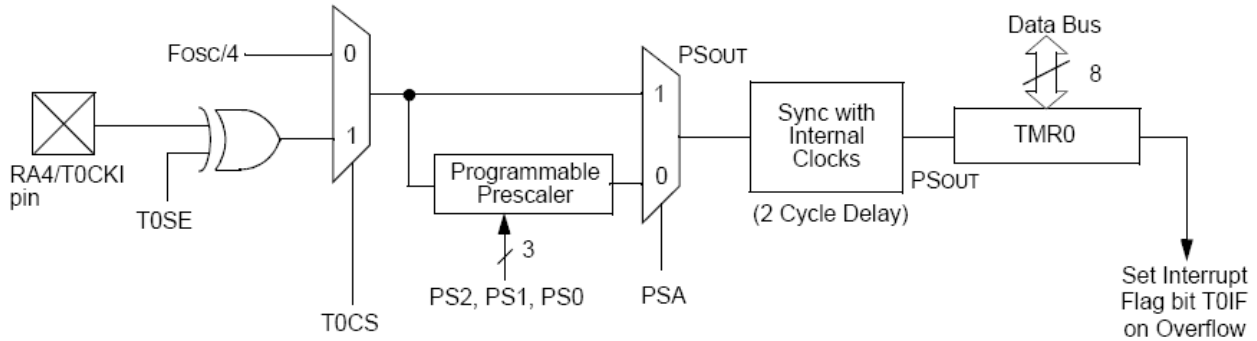
An important part of the interrupt process is the **stack.** Its basic role is to keep the value of program counter after a jump from the main program to an address of a subprogram. In order for a program to know how to go back to the point where it started from, it has to return the value of a program counter from a stack. When moving from a program to a subprogram, program counter is being pushed onto a stack. When executing instructions such as RETURN, RETLW or RETFIE which were executed at the end of a subprogram, program counter was taken from a stack so that program could continue from where it stopped before the interrupt. These operations of placing on and taking off from a program counter stack are called **PUSH** and **POP.**

However, the programmer has to keep the value of other registers used in the main program such as the status and work registers. This may be done as follows:

```
MOVWF  0Eh              ;push W and Status
MOVF   STATUS,W
MOVWF  0Fh

                        ; ISR(interrupt service routine)

MOVF   0Fh,W            ;pop W and Status
MOVWF  STATUS
SWAPF  0EH,F
SWAPF  0EH,W
```

2.6 TIMER0 (TMR0) module

FIGURE 5-1: TIMER0 BLOCK DIAGRAM



Note 1: T0CS, T0SE, PSA, PS2:PS0 (OPTION_REG<5:0>).
2: The prescaler is shared with Watchdog Timer (refer to Figure 5-2 for detailed block diagram).

TABLE 5-1: REGISTERS ASSOCIATED WITH TIMER0

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other RESETS
01h	TMR0	Timer0 Module Register								xxxx xxxx	uuuu uuuu
0Bh,8Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
85h	TRISA	—	—	—	PORTA Data Direction Register					---1 1111	---1 1111

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by Timer0.

REGISTER 2-2: OPTION REGISTER (ADDRESS 81h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

2.6.1 Delay with TIMER0:

A general formula for calculating the number of timer beats per second is as follows :

$$T = C/(4*P*R)$$

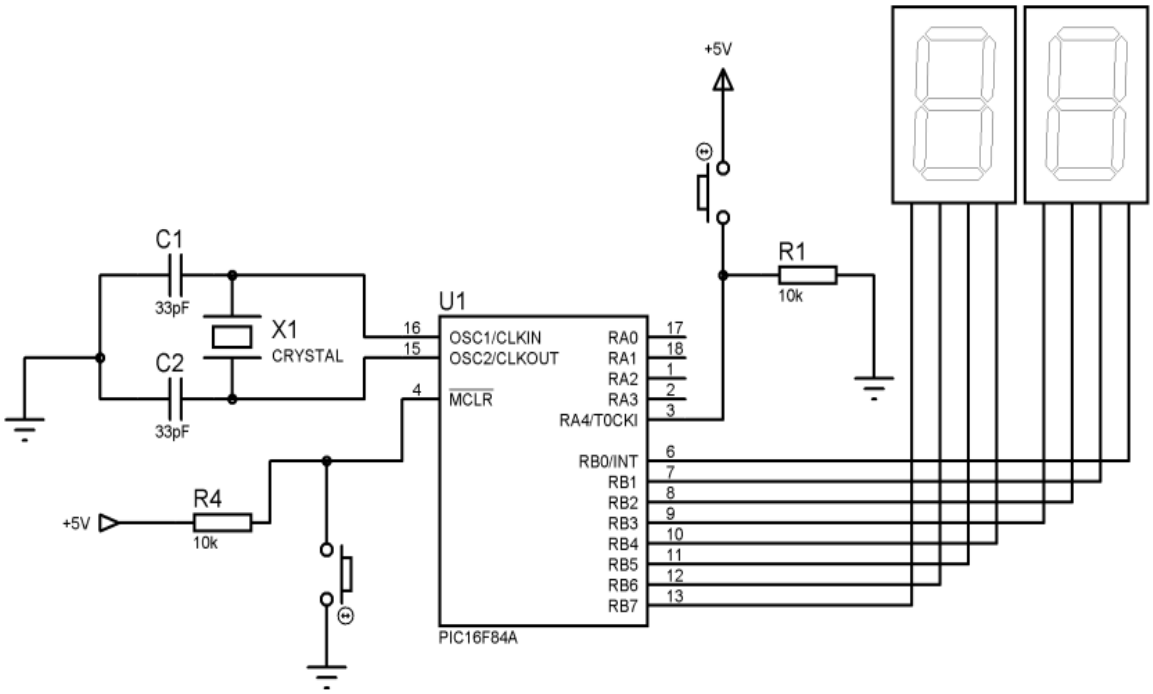
Where **T** is the number of clock beats per second ,**C** is the system clock speed in Hz , **P** is value stored in the prescaler , and **R** is the number of iterations counted in theTMR0 register. The range of both P and R in this formula is 1 to 256 .

```
Delay:
        MOVF    TMR0,W
        BTFSS   STATUS,Z
        GOTO    Delay
```

2.6.2 Counter mode:

The PIC16F84A can be programmed so hat portRA4/T0CKI is used to count events or pulses by initializing the Timer0 module as a counter. Without interrupts, the process requires the following preparatory steps:

- Port-A , line4 , (RA4/T0CKI) is defined for input.
- The Timer0 register (TMR0) is cleared.
- The OPTION register bits PSA and PS0:PS2 are initialized if the prescaler is to be used.
- The OPTION register bit T0SE is set so as to increment the count on the high-to-low transition of the port pin if the port source is active low .Otherwise the bit is cleared.
- The OPTION register bit T0CS is set to select action on the RA4/T0CKI pin.



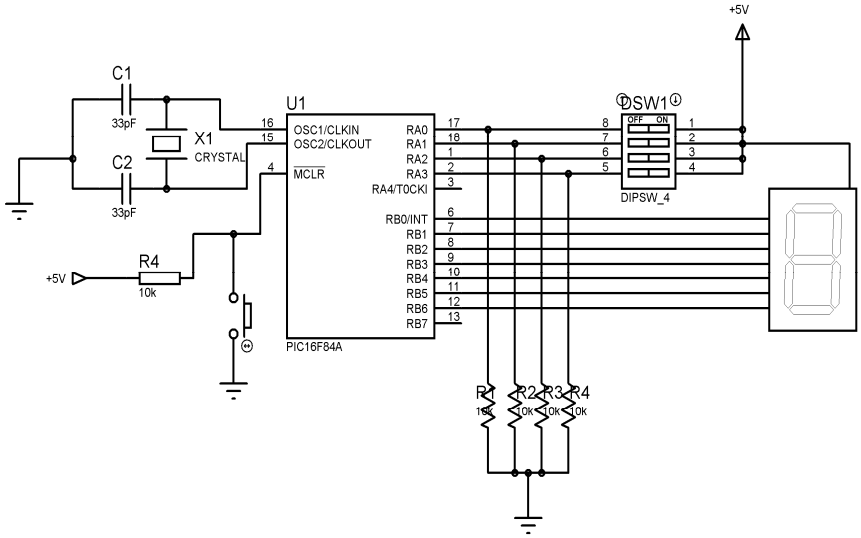

```
; Ex5 (Counter a) Counter on RA4/T0CKI from 00-FF
; No Interrupt is used here

          LIST   P=PIC16F84A
          #INCLUDE <P16F84A.INC>
          __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

          ORG    0X000
          GOTO   START
          ORG    0X004
          RETFIE

START     BSF    STATUS,RP0          ; BANK 1
          MOVLW  B'10111000'
          MOVWF  OPTION_REG          ; xx11 1xxx
          CLRF   TRISB               ; PORT B IS OUTPUT
          MOVLW  B'00011111'
          MOVWF  TRISA               ; PORT A IS INPUT
          BCF    STATUS,RP0          ;BANK0

          CLRF   TMR0
LOOP      MOVF   TMR0,W
          MOVWF  PORTB
          GOTO   LOOP
          END
```

```
; Ex5 (Counter_b) Counter on RA5/T0CKI Using Poling,
; (Bad implementation !)

          LIST   P=PIC16F84A
          #INCLUDE <P16F84A.INC>
          __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

          ORG    0X000
          GOTO   START
          ORG    0X004
          RETFIE

START     BSF    STATUS,RP0         ; BANK 1
          CLRF   TRISB              ; PORT B IS OUTPUT
          MOVLW  B'00011111'
          MOVWF  TRISA              ; PORT A IS INPUT
          BCF    STATUS,RP0         ; BANK0

          CLRF   PORTB
LOOP      BTFSS  PORTA,4
          GOTO   LOOP
          INCF   PORTB
          GOTO   LOOP
          END
```

```
; Ex5 (Counter_c) Counter on RA5/T0CKI Using Poling,
; (Good implementation !)
          LIST   P=PIC16F84A
          #INCLUDE <P16F84A.INC>
          __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

          ORG    0X000
          GOTO   START
          ORG    0X004
          RETFIE

START     BSF    STATUS,RP0         ; BANK 1
          CLRF   TRISB              ; PORT B IS oUTPUT
          MOVLW  B'00011111'
          MOVWF  TRISA              ; PORT A IS INPUT
          BCF    STATUS,RP0         ; BANK0

          CLRF   PORTB
LOOP      BTFSS  PORTA,4
          GOTO   LOOP
          INCF   PORTB              ; Key is pressed

LOOP1     BTFSC  PORTA,4
          GOTO   LOOP1
          GOTO   LOOP               ; Key is released

          END
```

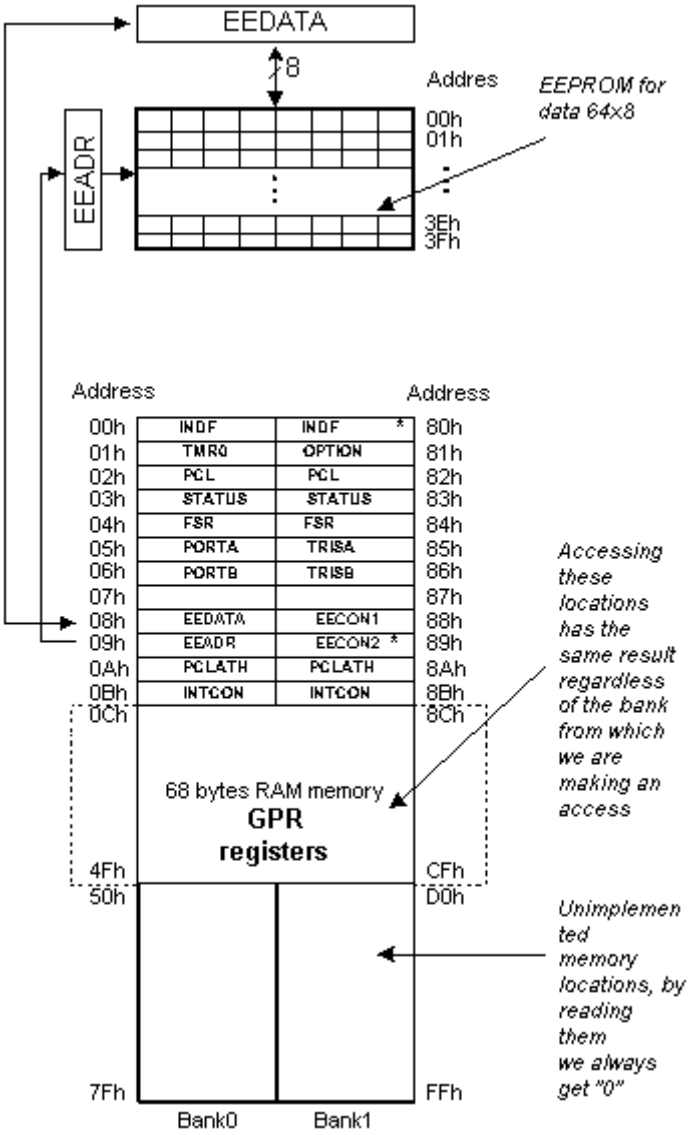
```
; Ex5 (Counter_d) Counter on RA5/T0CKI 0-99 twice,0-55 and restart

          LIST   P=PIC16F84A
          #INCLUDE <P16F84A.INC>
          __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

          CBLOCK  0X20
          TEMP
          DIG0
          DIG1
          DIG2
          ENDC
          ORG    0X000
          GOTO   START
          ORG    0X004
          RETFIE

START     BSF       STATUS,RP0      ; BANK 1
          MOVLW     B'10111000'
          BCF       OPTION_REG,6    ; 1-11 1000
          CLRF      TRISB           ; PORT B IS OUTPUT
          MOVLW     B'00011111'
          MOVWF     TRISA           ; PORT A IS INPUT
          BCF       STATUS,RP0      ; BANK0

          CLRF      TMR0
LOOP      MOVF      TMR0,W
          Call      BCD
          SWAPF     DIG1,W
          IORWF     DIG0,W
          MOVWF     PORTB
          GOTO      LOOP

BCD       MOVWF     TEMP
          CLRF      DIG1
          CLRF      DIG2
_100      MOVLW     .100            ;EXTRACT 100'S NUMBER
          SUBWF     TEMP,F          ;TEMP=TEMP-100
          BTFSS     STATUS,C
          GOTO      _10
          INCF      DIG2,F
          GOTO      _100
_10       MOVLW     .100            ;EXTRACT TEN'S NUMBER
          AddWF     TEMP,F
__10      MOVLW     .10
          SUBWF     TEMP,F          ;TEMP=TEMP-10
          BTFSS     STATUS,C
          GOTO      _1
          INCF      DIG1,F
          GOTO      __10
_1        MOVLW     .10
          ADDWF     TEMP,F
          MOVF      TEMP,W
          MOVWF     DIG0
          RETURN

          END
```

2.7 Look Up Tables




```
; Ex6 (LUT) An application for Luok up tables

                LIST   P=PIC16F84A
                #INCLUDE <P16F84A.INC>
                __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

                ORG    0X000
                GOTO   START
                ORG    0X004
                RETFIE

START           BSF      STATUS,RP0      ;BANK1
                CLRF     TRISB           ;ALL PORTB PINS ARE OUTPUT
                MOVLW    B'00011111'
                MOVWF    TRISA           ;PORTA PINS ARE INPUT
                BCF      STATUS,RP0

                MOVLW    C0h             ;0
                MOVWF    0Ch
                MOVLW    0F9h            ;1
                MOVWF    0Dh
                MOVLW    0A4h            ;2
                MOVWF    0Eh
                MOVLW    0B0h            ;3
                MOVWF    0Fh
                MOVLW    99h             ;4
                MOVWF    10h
                MOVLW    92h             ;5
                MOVWF    11h
                MOVLW    82h             ;6
                MOVWF    12h
                MOVLW    0F8h            ;7
                MOVWF    13h
                MOVLW    00h             ;8
                MOVWF    14h
                MOVLW    90h             ;9
                MOVWF    15h

LOOP            MOVLW    0Fh
                ANDWF    PORTA,W
                ADDLW    0Ch
                MOVWF    FSR
                MOVF     INDF,W
                MOVWF    PORTB
                GOTO     LOOP
                END
```

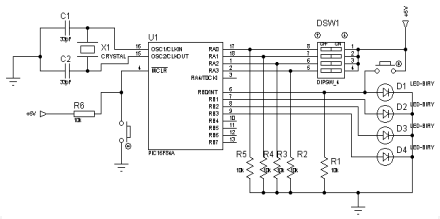
2.8 EEPROM utilization



The PIC16F84 and PIC16F84A contain 64 bytes of EEPROM data memory. This memory is both readable and writable during normal operation. It is not mapped in the register file space, but is indirectly addressed through the Special Function Registers EECON1, EECON2, EEDATA, and EEADR. The address of EEPROM memory starts at location 0x00 and extends to the maximum contained in the PIC, in this case, 0x3f. The following registers relate to EEPROM operations:

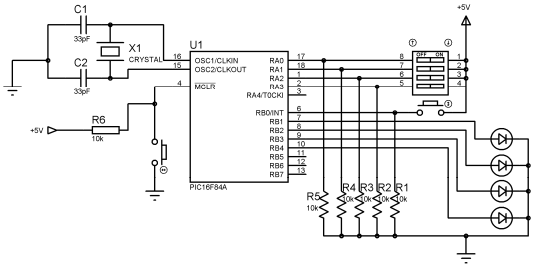
1. EEDATA holds the data byte to be read or written.
2. EEADR contains the EEPROM address to be accessed by the read or write operation.
3. EECON1 contains the control bits for EEPROM operations.
4 .EECON2 protects EEPROM memory from accidental access. This is not a physical register.

EECON1 Register				EEIF	WRERR	WREN	WR	RD




```
; Ex7 (EE_a) EEPROM handling application
            LIST   P=PIC16F84A
            #INCLUDE <P16F84A.INC>
            __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

            ORG    0X000
            GOTO   START
            ORG    0X004
            RETFIE

START       BSF    STATUS,RP0       ;BANK1
            CLRF   TRISB            ;ALL PORTB PINS ARE OUTPUT
            BSF    TRISB,0          ;Except B0 is input
            MOVLW  B'00011111'
            MOVWF  TRISA            ;PORTA PINS ARE INPUT

            BCF    STATUS,RP0
            MOVLW  07h
            MOvWF  EEADR            ;(1) address

LOOP0       BTFSS  PORTB,0
            GOTO   LOOP0
            MOVF   PORTA,W
            MOVWF  EEDATA           ;(2) data

            BSF    STATUS,RP0
            BCF    INTCON,GIE       ;(3) disable interrupts
            BCF    EECON1,EEIF      ;(4) clear EEIF
            BSF    EECON1,WREN      ;(5) enable writing
            MOVLW  55h              ;(6) keys
            MOVWF  EECON2
            MOVLW  0AAh
            MOVWF  EECON2
            BSF    EECON1,WR        ;(7) initialize writing
            BSF    INTCON,GIE       ;(8) enable interrupts

LOOP1       BTFSS  EECON1,EEIF      ;wait until writing is done
            GOTO   LOOP1

            BCF    STATUS,RP0
            CLRF   EEDATA

LOOP2       BTFSC  PORTB,0
            GOTO   LOOP2

            BSF    STATUS,RP0       ;EECON1 is at 88h
            BSF    EECON1,RD        ;Reading from EEPROM
            BCF    TRISB,0          ;B0 IS OUTPUT
            BCF    STATUS,RP0       ;EEDATA is at 08h
            MOVF   EEDATA,W
            MOVWF  PORTB

LOOP3       GOTO   LOOP3
            END
```

```
; Ex7 (EE_b) EEPROM handling application
            LIST   P=PIC16F84A
            #INCLUDE <P16F84A.INC>
            __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

            ORG    0X000
            GOTO   START
            ORG    0X004
            RETFIE

START       BSF    STATUS,RP0          ;BA
            MOVLW  B'00000001'         ;ALL PORTB PINS ARE OUTPUT
            MOVWF  TRISB               ;Except B0 is input
            MOVLW  B'00011111'
            MOVWF  TRISA               ;PORTA PINS ARE INPUT

            BCF    STATUS,RP0
            MOVLW  07h
            MOvWF  EEADR               ;(1) address

LOOP0       BTFSS  PORTB,0
            GOTO   LOOP0
            MOVF   PORTA,W
            MOVWF  EEDATA              ;(2) data

            BSF    STATUS,RP0
            BCF    INTCON,GIE          ;(3) disable interrupts
            BCF    EECON1,EEIF         ;(4) clear EEIF
            BSF    EECON1,WREN         ;(5) enable writing
            MOVLW  0X55                ;(6) keys
            MOVWF  EECON2
            MOVLW  0XAA
            MOVWF  EECON2
            BSF    EECON1,WR           ;(7) initialize writing
            BSF    INTCON,GIE          ;(8) enable interrupts

LOOP1       BTFSS  EECON1,EEIF         ;wait until writing is done
            GOTO   LOOP1
            BCF    EECON1,WREN
            BCF    STATUS,RP0
            CLRF   EEDATA

LOOP2       BTFSC  PORTB,0
            GOTO   LOOP2

            BSF    STATUS,RP0          ;EECON1 is at 88h
            BSF    EECON1,RD           ;Reading from EEPROM
            BCF    STATUS,RP0          ;EEDATA is at 08h
            RLF    EEDATA,W
            MOVWF PORTB

            GOTO  LOOP0
            END
```



2.9 Macros

Macros are a very useful element in assembly language. They could briefly be described as "user defined group of instructions which will enter assembler program where macro was called". It is possible to write a program even without using macros. But with their use written program is much more readable, especially if more programmers are working on the same program together. Macros have the same purpose as functions of higher program languages.

What's the difference between macro and subroutine?

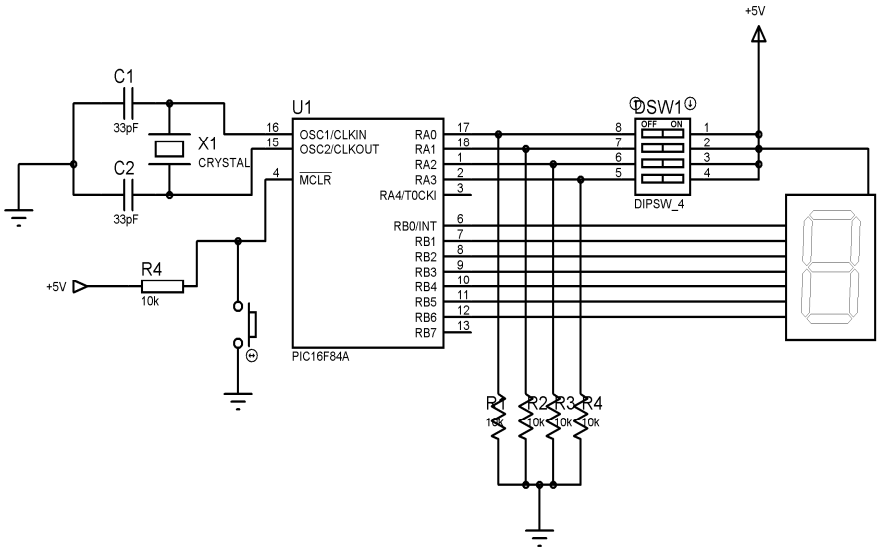
1) In execution stage, the subroutine puts the address of the called function in PC and hides the original value of PC (before calling) in the stack.

2) The macro in compilation process (not in execution), copies the macro in the place where it is called. It doesn't put address on stack.

3) The subroutine can be written or called in any place, but the macro must be declared before calling.

4) The subroutine takes more time in execution, but usually compiles to shorter programs

2.9.1 Using Macros internally



```
;This is an example for using MACROS internally in the
program.

                LIST   P=PIC16F84A
               #INCLUDE <P16F84A.INC>
               __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF
               ORG    0X000
               GOTO   START
               ORG    0X004
               RETFIE

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
BCDto7segTable Macro                    ;MACRO BEGINES HERE
               MOVLW     C0h        ;0
               MOVWF     0Ch
               MOVLW     0F9h       ;1
               MOVWF     0Dh
               MOVLW     0A4h       ;2
               MOVWF     0Eh
               MOVLW     0B0h       ;3
               MOVWF     0Fh
               MOVLW     99h        ;4
               MOVWF     10h
               MOVLW     92h        ;5
               MOVWF     11h
               MOVLW     82h        ;6
               MOVWF     12h
               MOVLW     0F8h       ;7
               MOVWF     13h
               MOVLW     00h        ;8
               MOVWF     14h
               MOVLW     90h        ;9
               MOVWF     15h
               ENDM                    ;MACRO ENDS HERE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

START          BSF    STATUS,RP0   ;BANK1
               CLRF   TRISB        ;ALL PORTB PINS ARE OUTPUT
               MOVLW  B'00011111'
               MOVWF  TRISA        ;PORTA PINS ARE INPUT
               BCF    STATUS,RP0

               BCDto7segTable      ;CALLING A MACRO

LOOP           MOVLW  0Fh
               ANDWF  PORTA,W
               ADDLW  0Ch
               MOVWF  FSR
               MOVF   INDF,W
               MOVWF  PORTB
               GOTO   LOOP
               END
```

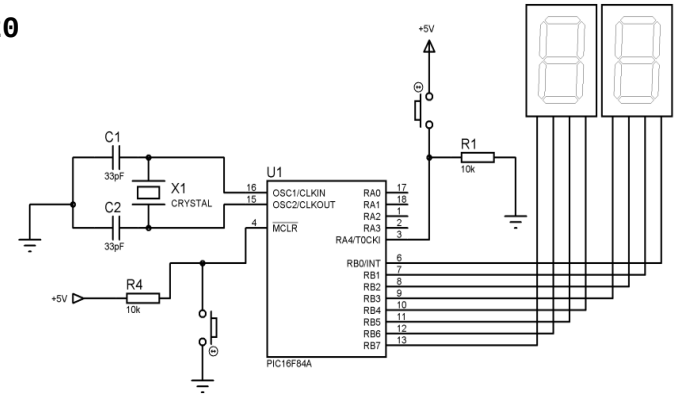
2.9.2 Using Macros externally:

```
;This is an example for using MACROS externally in the program.

          LIST   P=PIC16F84A
          #INCLUDE <P16F84A.INC>
          #INCLUDE <MyLibrary.INC>
          __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF
          ORG    0X000
          GOTO   START
          ORG    0X004
          RETFIE



START     BSF    STATUS,RP0      ;BANK1
          CLRF   TRISB           ;ALL PORTB PINS ARE OUTPUT
          MOVLW  B'00011111'
          MOVWF  TRISA           ;PORTA PINS ARE INPUT
          BCF    STATUS,RP0

          BCDto7segTable         ;CALLING A MACRO

LOOP      MOVLW  0Fh
          ANDWF  PORTA,W
          ADDLW  0Ch
          MOVWF  FSR
          MOVF   INDF,W
          MOVWF  PORTB
          GOTO   LOOP
          END
```

Mylibrary.INC

```
BCDto7segTable   Macro
                 MOVLW  0C0h     ;0
                 MOVWF  0Ch
                 MOVLW  0F9h     ;1
                 MOVWF  0Dh
                 MOVLW  0A4h     ;2
                 MOVWF  0Eh
                 MOVLW  0B0h     ;3
                 MOVWF  0Fh
                 MOVLW  99h ;4
                 MOVWF  10h
                 MOVLW  92h ;5
                 MOVWF  11h
                 MOVLW  82h ;6
                 MOVWF  12h
                 MOVLW  0F8h     ;7
                 MOVWF  13h
                 MOVLW  00h ;8
                 MOVWF  14h
                 MOVLW  90h ;9
                 MOVWF  15h
                 ENDM
```

```
;Ex8 (Macro) Decimal Counter on RA5/T0CKI as Ex5d
            LIST   P=PIC16F84A
            #INCLUDE <P16F84A.INC>
            #INCLUDE <ECOM4315.INC>
            __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

            CBLOCK     0X20
            TEMP
            DIG0
            DIG1
            DIG2
            ENDC

            ORG     0X000
            GOTO    START
            ORG     0X004
            RETFIE
```




```
START       BSF         STATUS,RP0          ; BANK 1
            MOVLW       B'10111000'
            BCF         OPTION_REG,6        ; 1-11 1000
            CLRF        TRISB               ; PORT B IS oUTPUT
            MOVLW       B'00011111'
            MOVWF       TRISA         ; PORT A IS INPUT
            BCF         STATUS,RP0          ; BANK0

            CLRF        TMR0
LOOP        MOVF        TMR0,W
            BCD         TEMP,DIG0,DIG1,DIG2
            SWAPF       DIG1,W
            IORWF       DIG0,W
            MOVWF       PORTB
            GOTO        LOOP
            END
```

```
; file name is ECOM5315.INC
BCD         Macro   Temp,DIG0,DIG1,DIG2
            local   _100, _10, __10, _1
            MOVWF   TEMP
            CLRF    DIG1
            CLRF    DIG2
_100        MOVLW   .100        ;EXTRACT 100'S NUMBER
            SUBWF   TEMP,F  ;TEMP=TEMP-100
            BTFSS   STATUS,C
            GOTO    _10
            INCF    DIG2,F
            GOTO    _100

_10         MOVLW   .100        ;EXTRACT TEN'S NUMBER
            AddWF   TEMP,F

__10        MOVLW   .10
            SUBWF   TEMP,F    ;TEMP=TEMP-10
            BTFSS   STATUS,C
            GOTO    _1
            INCF    DIG1,F
            GOTO    __10

_1          MOVLW   .10
            ADDWF   TEMP,F
            MOVF    TEMP,W
            MOVWF   DIG0
            ENDM
```

2.10 The hexadecimal object file format for PIC Microcontroller

General Record Format:

Record Mark ':'	Load Record Length	Offset	Record Type	Information or Data	Chick Sum
1-byte	1-byte	2-bytes	1-byte	n-bytes	1-byte

Record Mark: each record begins with a Record Mark field containing 03AH, the ASCII code for the colon ':' character.

Record Length: The number of bytes of information or data.

Offset: By the offset we can arrive to the address.

Record Type: It is used to interpret the remaining information within the record. The encoding for all the current record types is:

Record Type	Description	Information or Data Digits
00	Data Record	x
01	End of File Record	0
02	Extended Segment Address Record	4
03	Start Segment Address Record	8
04	Extended Linear Address Record	4
05	Start Linear Address Record	8

Information or Data: It consists of zero or more bytes encoded as pairs of hexadecimal digits. The interpretation of this field depends on the Record Type field.

Chick Sum: This field contains the check sum on **the Record length**, **Load Offset, Record Type**, and **Information or Data**. Therefore, the sum of all the ASCII pairs in a record after converting to binary, from the Record length field to and including the Chick Sum field, is zero.

.HEX File of Ex1

```
:020000040000FA
:020000000528D1
:0800080009008316051006141F
:0800100083120608850009288F
:02400E00F13F80
:00000001FF
```

Record Mark	Record Length	Offset	Record Type	Info. or Data	Chick Sum
:	02	0000	04	0000	FA
:	02	0000	00	0528	D1
:	08	0008	00	0900831605100614	1F
:	08	0010	00	8312060885000928	8F
:	02	400E	00	F13F	80
:	00	0000	01		FF

Modeling the program memory in PIC Microcontrollers

(Hexa notation)

Address Address	content	
0000	28	05
0001	xx	xx
0002	xx	xx
0003	xx	xx
0004	00	09
0005	16	83
0006	10	05
0007	14	06
0008	12	83
0009	08	06
000A	00	85
000B	28	09
…		
…		
…		
1FFF	xx	xx
2007	3F	F1

(Binary notation)

Address	content													
0000000000000	1	0	1	0	0	0	0	0	0	0	0	1	0	1
0000000000001	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0000000000010	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0000000000011	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0000000000100	0	0	0	0	0	0	0	0	0	0	1	0	0	1
0000000000101	0	1	0	1	1	0	1	0	0	0	0	0	1	1
0000000000110	0	1	0	0	0	0	0	0	0	0	0	1	0	1
0000000000111	0	1	0	1	0	0	0	0	0	0	0	1	1	0
0000000001000	0	1	0	0	1	0	1	0	0	0	0	0	1	1
0000000001001	0	0	1	0	0	0	0	0	0	0	0	1	1	0
0000000001010	0	0	0	0	0	0	1	0	0	0	0	1	0	1
0000000001011	1	0	1	0	0	0	0	0	0	0	1	0	0	1
…														
…														
…														
1111111111111														
10000000000111	1	1	1	1	1	1	1	1	1	1	0	0	0	1

4-1 Introduction to C language

The ‘C’ programming language allows applications to be written using syntax whose meaning is a little easier to understand than assembly code. Programs in C are converted into assembly language by a compiler, and assembled into machine code, in a two-stage process.

1- Compile to assembly level code.
2- Assemble link to machine code.

The process of compiling, along with the files used and generated, is illustrated in Figure 4-1 The main program, the C source file, is written in the C language, in a file with the extension .c.

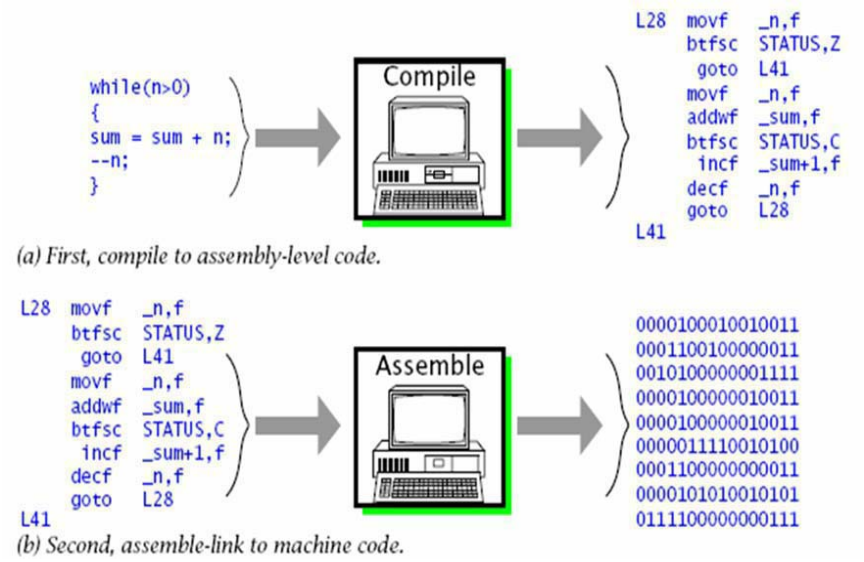


Figure 4-1
Process of compiling the C source file

‘C’ is the high-level language of choice for PIC microcontrollers. A range of different development systems and compilers are available, but most use the same basic syntax defined as ANSI (American National Standards Institute) C. Assembly language is syntax which is unique to each type of processor, while C provides a common language for all MCU(Microcontroller unit) types.

4-2 Introduction to mikroC

mikroC is a powerful, feature rich development tool for PICmicros. It is designed to provide the programmer with the easiest possible solution for developing applications for embedded systems, without compromising performance or control.

PIC and C fit together well: PIC is the most popular 8-bit chip in the world, used in a wide variety of applications, and C, prized for its efficiency, is the natural choice for developing embedded systems. mikroC provides a successful match featuring highly advanced IDE, ANSI compliant compiler, broad set of hardware libraries, comprehensive documentation, and plenty of ready-to-run examples.

mikroC allows you to quickly develop and deploy complex applications:

- Write your C source code using the built-in Code Editor (Code and Parameter Assistants, Syntax Highlighting, Auto Correct, Code Templates, and more…)
- Use the included mikroC libraries to dramatically speed up the development: data acquisition, memory, displays, conversions, communications… Practically all P12, P16, and P18 chips are supported.
- Monitor your program structure, variables, and functions in the Code Explorer.
- Generate commented, human-readable assembly, and standard HEX compatible with all programmers.
- Inspect program flow and debug executable logic with the integrated Debugger.
- Get detailed reports and graphs: RAM and ROM map, code statistics, assembly listing, calling tree, and more…
- We have provided plenty of examples for you to expand, develop, and use as building bricks in your projects. Copy them entirely if you deem fit – that's why we included them with the compiler.

mikroC is a user-friendly and intuitive environment:

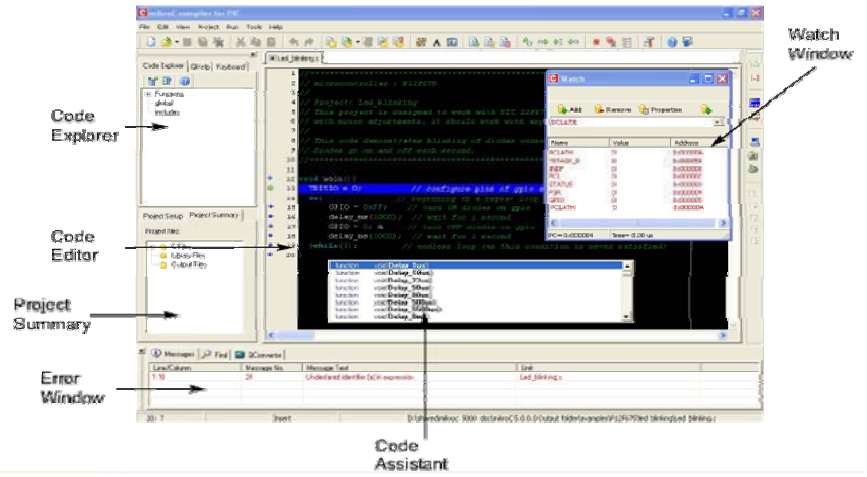


Figure 4-2
mikroC IDE

- The **Code Explorer** (with Keyboard shortcut browser and Quick Help browser) is at your disposal for easier project management.
- The **Error Window** displays all errors detected during compiling and linking.
- The source-level **Debugger** lets you debug executable logic step-by-step by watching the program flow.
- The **New Project Wizard** is a fast, reliable, and easy way to create a project.
- Help files are syntax and context sensitive.
- As with any modern Windows application, you may customize the layout of mikroC to best suit your needs. See (Figure 4-2).

4-2 Building the First Program (Flasher)

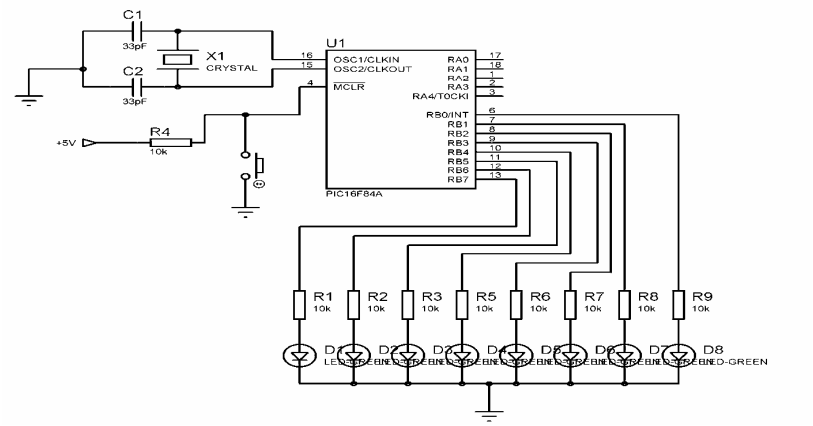


Figure 4-3
Flasher on PORTB

```
/*This program is a simple program to make a flasher wave on PORTB of the PIC16F84A,
using an infinite loop to make sure that the wave still blinking on PORTB, and using a
delay of 0.5 sec between every flashing led*/

void main()
{
 TRISB = 0;
 PORTB = 0;


 while(1) {
            PORTB =  ~PORTB;
            Delay_ms(1000);
           }
}
```

Notes:

- In MicroC, we didn't need to initialize the banks. We can deal with the each ports directly.
- The line command *PORTB = ~PORTB* is used to get the complement status of the previous status in PORTB to apply the idea of the flasher.
- The function *Delay_ms* is used to make delay.

To make flasher on one bit only we can apply one of the following ways:

1-

```
PORTB = (~ PORTB & 0b00000001) | (PORTB & 0b11111110)
```

Notes:

This line is used to apply the XOR operation.
| → It is a bit wise operation, dealing with the number bit by bit.
|| → It is dealing with the number as one unit.

2-

```
PORTB= PORTB & 0b11111110
Delay_ms(1000);
PORTB= PORTB | 0b00000001
Delay_ms(1000);
```

The procedures to write this is program in mikroC program:

1-From project select New Project

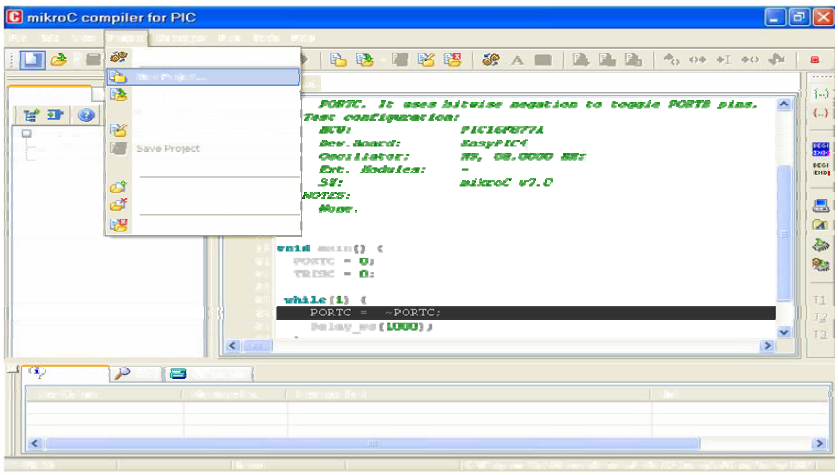


Figure 4-4
New Project.

2- Enter Project Name, Project Path, Device, Clock and Configurations.

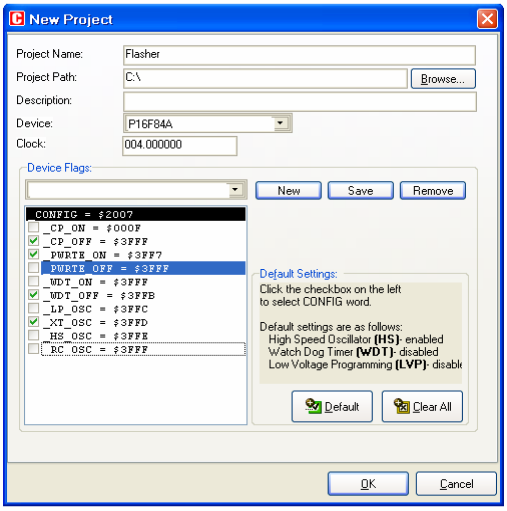


Figure 4-5

Project Name, Project Path, Device, Clock and Configurations.

3-Write the program in Code Editor

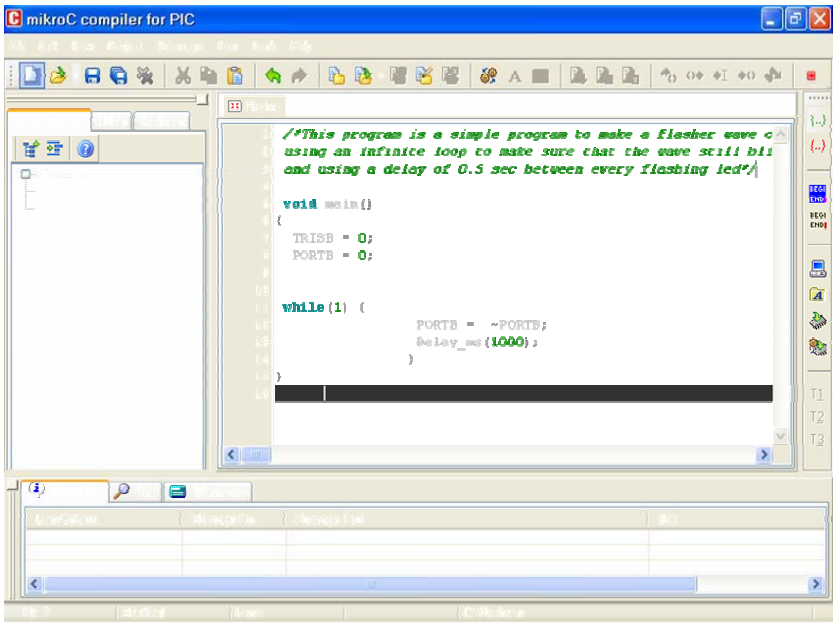

mikroC compiler for PIC
/*This program is a simple program to make a flasher wave c
using an infinite loop to make sure that the wave still bli
and using a delay of 0.5 sec between every flashing led*/
void main()
{
TRISB = 0;
PORTB = 0;
while(1) {
PORTB = ~PORTB;
Delay_ms(1000);
}
}

Figure 4-6

Program Writing.

4- From Project enter Build

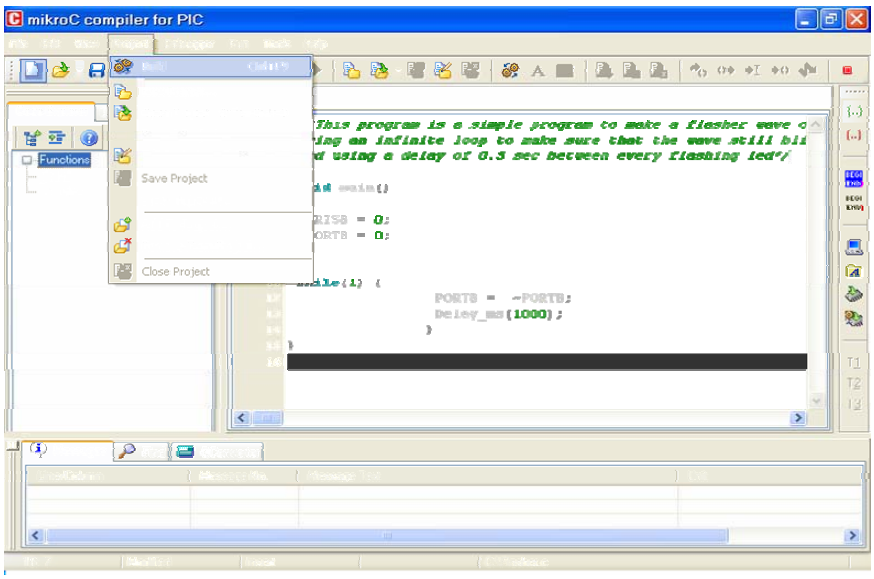


Figure 4-7
Program Building.

5- After Build the following message well appears.

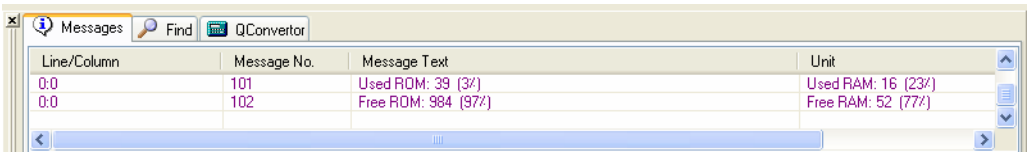


Figure 4-8
Message after Building.

4-3 IQAMA Application

The idea of this program to build an application to be used to display the time remaining to the Iqama after each Athan of the Muslim's fife pray times. This application can be implemented as shown in (Figure 4-8)

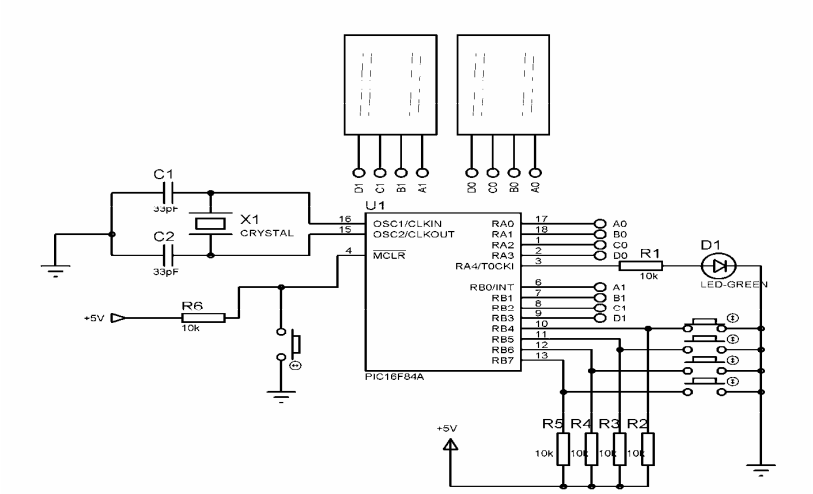


Figure 4-9

IQAMA application.

```
//Iqama Application

int i, j;
void interrupt()
{
 INTCON=INTCON & 0b11111110;
 if ((PORTB & 0b00010000)==0b0) i=10;
 if ((PORTB & 0b00100000)==0b0) i=15;
 if ((PORTB & 0b01000000)==0b0) i=20;
}

void main()
{
 INTCON   = 0b10001000;
 TRISA    = 0;
 PORTA    = 0;
 TRISB    = 0xF0;
 PORTB    = 0;
 while(1)
   {
 while(i>0)
       {
         j = i % 10;
         PORTA = j;
         j = (i / 10) % 10;
         PORTB = j;
         Delay_us(995840);
         i-- ;
       }
   PORTA =0;
   PORTB =0;
  }
}
```

Notes:

- The idea of the program is to determine the time interval between each Athan and Iqama. Each switch is used to determine the interval of a specific Athan. The time appeared on the two 7-segment.
- In low level language, the counter is a memory location. But in high level language to make a counter we just need to declare variable.
- Port A was set to be output. $B_0 \rightarrow B_3$ output and $B_4 \rightarrow B_7$ were adjusted to be input.
- The line command *if ((PORTB & 0b00010000)==0b0) i=10;* is used to determine the time or the count of each switch to represent the interval of a specific Athan.
- The code inside the while is used to send the LSB and MSB on PORTA, PORTB respectively.
- In Micro C code, Retfie is not important to be written.

EEPROM Library

EEPROM data memory is available with a number of PICmicros. mikroC includes library for comfortable work with EEPROM.

Library Routines

- Eeprom_Read
- Eeprom_Write

Eeprom_Read

Prototype	`unsigned short Eeprom_Read(unsigned int address);`
Returns	Returns byte from specified address.
Description	Reads `data` from specified `address`. Parameter `address` is of integer type, which means it supports MCUs with more than 256 bytes of EEPROM.
Requires	Requires EEPROM module. Ensure minimum 20ms delay between successive use of routines `Eeprom_Write` and `Eeprom_Read`. Although PIC will write the correct value, `Eeprom_Read` might return an undefined result.

Example	`unsigned short take;`
	`...`
	`take = Eeprom_Read(0x3F);`

Eeprom_Write

Prototype	`void Eeprom_Write(unsigned int address, unsigned short data);`
Returns	Nothing.
Description	Writes `data` to specified `address`. PParameter `address` is of integer type, which means it supports MCUs with more than 256 bytes of

	EEPROM. Be aware that all interrupts will be disabled during execution of `Eeprom_Write` routine (GIE bit of INTCON register will be cleared). Routine will restore previous state of this bit on exit.
Requires	Requires EEPROM module. Ensure minimum 20ms delay between successive use of routines `Eeprom_Write` and `Eeprom_Read`. Although PIC will write the correct value, `Eeprom_Read` might return an undefined result.
Example	`Eeprom_Write(0x32, 19);`

Library Example

The example demonstrates using EEPROM Library.

```
unsigned short i = 0, j = 0;

void main() {
  PORTB = 0;
  TRISB = 0;

  j = 4;
  for (i = 0; i < 20u; i++)
    EEprom_Write(i, j++);

  for (i = 0; i < 20u; i++) {
    PORTB = Eeprom_Read(i);
    Delay_ms(500);
  }
}//~!
```

LCD Library

mikroC provides a library for communication with LCDs (with HD44780 compliant controllers) through the 4-bit interface. An example of LCD connections is given on the schematic at the bottom of this page.

For creating a custom set of LCD characters use LCD Custom Character Tool.

Note: mikroElektronika's development system based initialization routines are included in Code Templates.

Note: Only `Lcd_Config` and `Lcd_Init` routines use the RW pin (RW pin is configured as output and set to zero). If the user needs this pin for other purposes, it can be reconfigured after `Lcd_Config` or `Lcd_Init` call.

Library Routines

- Lcd_Config
- Lcd_Init
- Lcd_Out
- Lcd_Out_Cp
- Lcd_Chr
- Lcd_Chr_Cp
- Lcd_Cmd

Lcd_Config

Prototype	`void Lcd_Config(unsigned short *port, unsigned short RS, unsigned short EN, unsigned short WR, unsigned short D7, unsigned short D6, unsigned short D5, unsigned short D4);`
Returns	Nothing.
Description	Initializes LCD at `port` with pin settings you specify: parameters `RS`, `EN`, `WR`, `D7` .. `D4` need to be a combination of values 0–7 (e.g. 3,6,0,7,2,1,4).
Requires	Nothing.
Example	`Lcd_Config(&PORTD, 0, 1, 7, 5, 4, 3, 2);`

Lcd_Init

Prototype	`void Lcd_Init(unsigned short *port);`
Returns	Nothing.

Description	Initializes LCD at `port` with default pin settings (see the connection scheme at the end of the chapter):
	`D7 → port.7`
	`D6 → port.6`
	`D5 → port.5`
	`D4 → port.4`
	`E → port.3`

	RS → port.2 RW → port.0
Requires	Nothing.
Example	`Lcd_Init(&PORTB);`

Lcd_Out

Prototype	`void Lcd_Out(unsigned short row, unsigned short col, char *text);`
Returns	Nothing.
Description	Prints `text` on LCD at specified row and column (parameters `row` and `col`). Both string variables and literals can be passed as `text`.
Requires	Port with LCD must be initialized. See Lcd_Config or Lcd_Init.
Example	Print “Hello!” on LCD at line 1, char 3: `Lcd_Out(1, 3, "Hello!");`

Lcd_Out_Cp

Prototype	`void Lcd_Out_Cp(char *text);`
Returns	Nothing.
Description	Prints `text` on LCD at current cursor position. Both string variables and literals can be passed as `text`.
Requires	Port with LCD must be initialized. See Lcd_Config or Lcd_Init.
Example	Print “Here!” at current cursor position: `Lcd_Out_Cp("Here!");`

Lcd_Chr

Prototype	`void Lcd_Chr(unsigned short row, unsigned short col, char character);`
Returns	Nothing.
Description	Prints `character` on LCD at specified row and column (parameters `row` and `col`). Both variables and literals can be passed as `character`.
Requires	Port with LCD must be initialized. See Lcd_Config or Lcd_Init.
Example	Print “i” on LCD at line 2, char 3: `Lcd_Chr(2, 3, 'i');`

Lcd_Chr_Cp

Prototype	`void Lcd_Chr_Cp(char character);`
Returns	Nothing.
Description	Prints `character` on LCD at current cursor position. Both variables and literals can be passed as `character`.
Requires	Port with LCD must be initialized. See Lcd_Config or Lcd_Init.
Example	Print “e” at current cursor position: `Lcd_Chr_Cp('e');`

Lcd_Cmd

Prototype	`void Lcd_Cmd(unsigned short command);`
Returns	Nothing.
Description	Sends `command` to LCD. You can pass one of the predefined constants to the function. The complete list of available commands is below.
Requires	Port with LCD must be initialized. See Lcd_Config or Lcd_Init.
Example	Clear LCD display: `Lcd_Cmd(Lcd_Clear);`

Available LCD Commands

LCD Command	Purpose
LCD_FIRST_ROW	Move cursor to 1st row
LCD_SECOND_ROW	Move cursor to 2nd row
LCD_THIRD_ROW	Move cursor to 3rd row
LCD_FOURTH_ROW	Move cursor to 4th row
LCD_CLEAR	Clear display
LCD_RETURN_HOME	Return cursor to home position, returns a shifted display to original position. Display data RAM is unaffected.
LCD_CURSOR_OFF	Turn off cursor
LCD_UNDERLINE_ON	Underline cursor on
LCD_BLINK_CURSOR_ON	Blink cursor on
LCD_MOVE_CURSOR_LEFT	Move cursor left without changing display data RAM
LCD_MOVE_CURSOR_RIGHT	Move cursor right without changing display data RAM
LCD_TURN_ON	Turn LCD display on

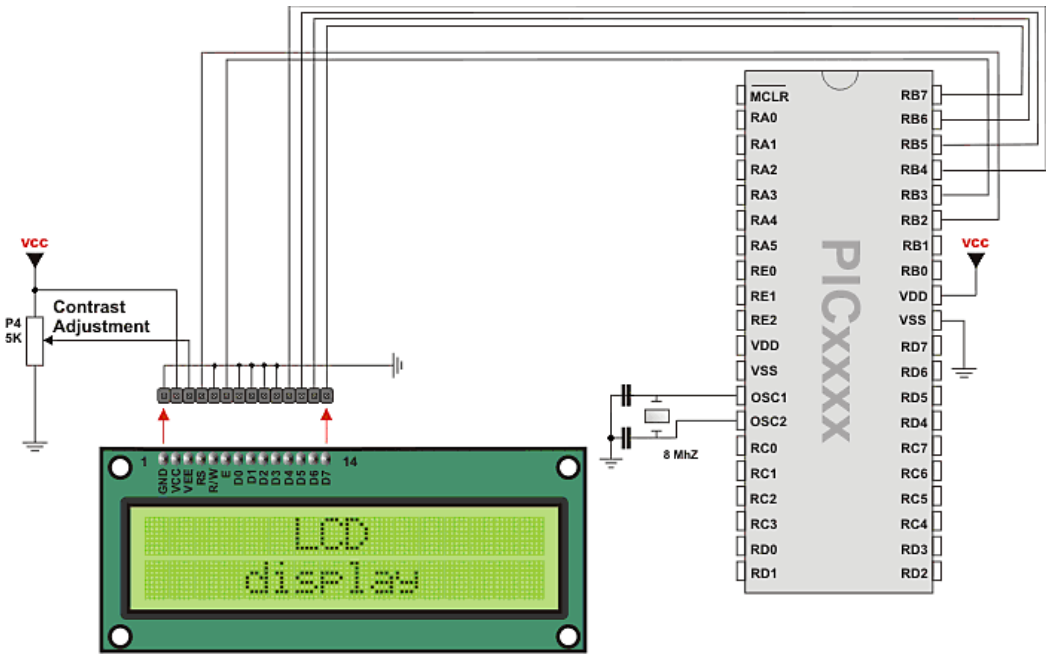
LCD_TURN_OFF	Turn LCD display off
LCD_SHIFT_LEFT	Shift display left without changing display data RAM
LCD_SHIFT_RIGHT	Shift display right without changing display data RAM

Library Examples

Default Pin Configuration

Use Lcd_Init for default pin settings (see the first figure below).

```
char *text = "mikroElektronika";

void main() {
  TRISB = 0;                // PORTB is output
  Lcd_Init(&PORTB);         // Initialize LCD connected to PORTB
  Lcd_Cmd(Lcd_CLEAR);       // Clear display
  Lcd_Cmd(Lcd_CURSOR_OFF);  // Turn cursor off
  Lcd_Out(1, 1, text);      // Print text to LCD, 2nd row, 1st column
}//~!
```



LCD HW connection by default initialization (using Lcd_Init)

LCD8 (8-bit interface) Library

mikroC provides a library for communication with LCDs (with HD44780 compliant controllers) through the 8-bit interface. An example of LCD connections is given on the schematic at the bottom of this page.

For creating a custom set of LCD characters use LCD Custom Character Tool.

Note: mikroElektronika's development system based initialization routines are included in Code Templates.

Note: Only `Lcd8_Config` and `Lcd8_Init` routines use the RW pin (RW pin is configured as output and set to zero). If the user needs this pin for other purposes, it can be reconfigured after the `Lcd8_Config` or `Lcd8_Init` call.

Library Routines

- Lcd8_Config
- Lcd8_Init
- Lcd8_Out
- Lcd8_Out_Cp
- Lcd8_Chr
- Lcd8_Chr_Cp
- Lcd8_Cmd

Lcd8_Config

Prototype	`void Lcd8_Config(unsigned short *ctrlport, unsigned short *dataport, unsigned short RS, unsigned short EN, unsigned short WR, unsigned short D7, unsigned short D6, unsigned short D5, unsigned short D4, unsigned short D3, unsigned short D2, unsigned short D1, unsigned short D0);`
Returns	Nothing.
Description	Initializes LCD at Control port (`ctrlport`) and Data port (`dataport`) with pin settings you specify. Parameters `RS`, `EN`, `WR` need to be in range 0–7; Parameters `D7` .. `D0` need to be a combination of values 0–7 (e.g. 3,6,5,0,7,2,1,4).
Requires	Nothing.
Example	`Lcd8_Config(&PORTC,&PORTD,0,1,2,6,5,4,3,7,1,2,0);`

Lcd8_Init

Prototype	`void Lcd8_Init(unsigned short *portctrl, unsigned short *portdata);`
Returns	Nothing.
Description	Initializes LCD at Control port (`portctrl`) and Data port (`portdata`) with default pin settings (see the connection scheme at the end of the chapter):

	E → ctrlport.6 RS → ctrlport.4 R/W → ctrlport.5 D7 → dataport.7 D6 → dataport.6 D5 → dataport.5 D4 → dataport.4 D3 → dataport.3 D2 → dataport.2 D1 → dataport.1 D0 → dataport.0
Requires	Nothing.
Example	`Lcd8_Init(&PORTB, &PORTC);`

Lcd8_Out

Prototype	`void Lcd8_Out(unsigned short row, unsigned short col, char *text);`
Returns	Nothing.
Description	Prints `text` on LCD at specified row and column (parameter `row` and `col`). Both string variables and literals can be passed as `text`.
Requires	Ports with LCD must be initialized. See Lcd8_Config or Lcd8_Init.
Example	Print “Hello!” on LCD at line 1, char 3: `Lcd8_Out(1, 3, "Hello!");`

Lcd8_Out_Cp

Prototype	`void Lcd8_Out_Cp(char *text);`
Returns	Nothing.
Description	Prints `text` on LCD at current cursor position. Both string variables and literals can be passed as `text`.
Requires	Ports with LCD must be initialized. See Lcd8_Config or Lcd8_Init.
Example	Print “Here!” at current cursor position: `Lcd8_Out_Cp("Here!");`

Lcd8_Chr

Prototype	`void Lcd8_Chr(unsigned short row, unsigned short col, char character);`
Returns	Nothing.

Description	Prints `character` on LCD at specified row and column (parameter `row` and `col`). Both variables and literals can be passed as `character`.
Requires	Ports with LCD must be initialized. See Lcd8_Config or Lcd8_Init.
Example	Print "i" on LCD at line 2, char 3: `Lcd8_Out(2, 3, 'i');`

Lcd8_Chr_Cp

Prototype	`void Lcd8_Chr_Cp(char character);`
Returns	Nothing.
Description	Prints `character` on LCD at current cursor position. Both variables and literals can be passed as `character`.
Requires	Ports with LCD must be initialized. See Lcd8_Config or Lcd8_Init.
Example	Print "e" at current cursor position: `Lcd8_Chr_Cp('e');`

Lcd8_Cmd

Prototype	`void Lcd8_Cmd(unsigned short command);`
Returns	Nothing.
Description	Sends `command` to LCD. You can pass one of the predefined constants to the function. The complete list of available commands is below.
Requires	Ports with LCD must be initialized. See Lcd8_Config or Lcd8_Init.
Example	Clear LCD display: `Lcd8_Cmd(Lcd_Clear);`

Available LCD Commands

LCD Command	Purpose
`LCD_FIRST_ROW`	Move cursor to 1st row
`LCD_SECOND_ROW`	Move cursor to 2nd row
`LCD_THIRD_ROW`	Move cursor to 3rd row
`LCD_FOURTH_ROW`	Move cursor to 4th row
`LCD_CLEAR`	Clear display
`LCD_RETURN_HOME`	Return cursor to home position, returns a shifted display to original position. Display data RAM is unaffected.

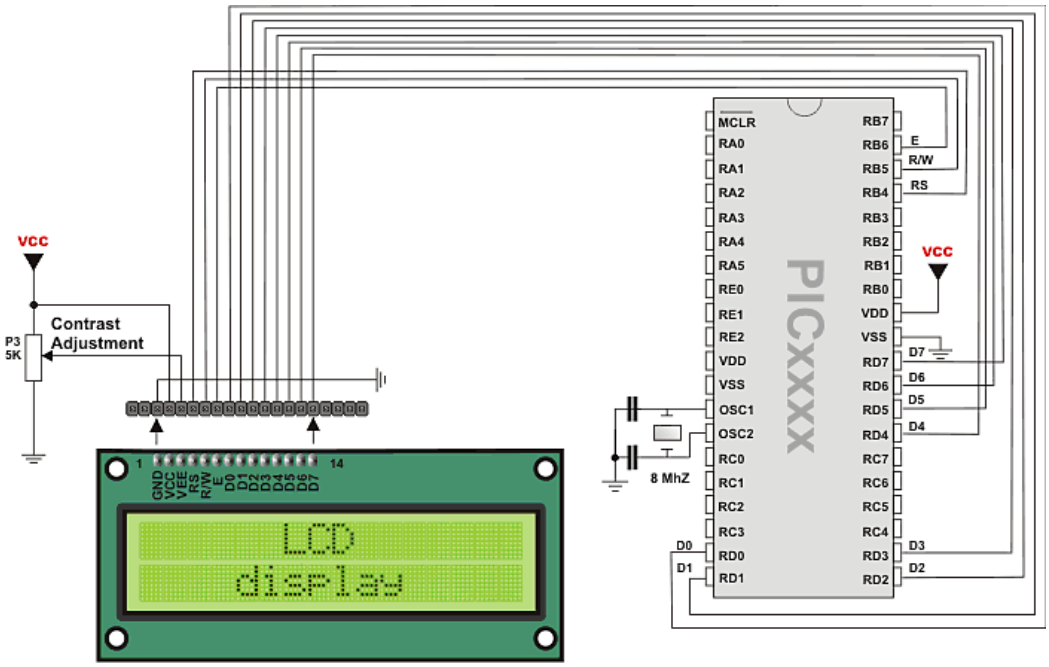
LCD_CURSOR_OFF	Turn off cursor
LCD_UNDERLINE_ON	Underline cursor on
LCD_BLINK_CURSOR_ON	Blink cursor on
LCD_MOVE_CURSOR_LEFT	Move cursor left without changing display data RAM
LCD_MOVE_CURSOR_RIGHT	Move cursor right without changing display data RAM
LCD_TURN_ON	Turn LCD display on
LCD_TURN_OFF	Turn LCD display off
LCD_SHIFT_LEFT	Shift display left without changing display data RAM
LCD_SHIFT_RIGHT	Shift display right without changing display data RAM

Library Examples

Default Pin Configuration

Use `Lcd8_Init` for default pin settings (see the first figure below).

```
char *text = "mikroElektronika";

void main() {
  TRISB = 0;                  // PORTB is output
  TRISC = 0;                  // PORTC is output
  Lcd8_Init(&PORTB, &PORTC);  // Initialize LCD at PORTB and PORTC
  Lcd8_Cmd(LCD_CURSOR_OFF);   // Turn off cursor
  Lcd8_Out(1, 1, text);       // Print text on LCD
}
```



LCD HW connection by default initialization (using `Lcd8_Init`)

Keypad Library

mikroC provides library for working with 4x4 keypad; routines can also be used with 4x1, 4x2, or 4x3 keypad. Check the connection scheme at the end of the topic.

Library Routines

- Keypad_Init
- Keypad_Read
- Keypad_Released

Keypad_Init

Prototype	`void Keypad_Init(unsigned *port);`
Returns	Nothing.
Description	Initializes `port` to work with keypad. The function needs to be called before using other routines from Keypad library.
Requires	Nothing.
Example	`Keypad_Init(&PORTB);`

Keypad_Read

Prototype	`unsigned short Keypad_Read(void);`
Returns	1..16, depending on the key pressed, or 0 if no key is pressed.
Description	Checks if any key is pressed. Function returns 1 to 16, depending on the key pressed, or 0 if no key is pressed.
Requires	Port needs to be appropriately initialized; see Keypad_Init.
Example	`kp = Keypad_Read();`

Keypad_Released

Prototype	`unsigned short Keypad_Released(void);`
Returns	1..16, depending on the key.
Description	Call to `Keypad_Released` is a blocking call: function waits until any key is pressed *and* released. When released, function returns 1 to 16, depending on the key.
Requires	Port needs to be appropriately initialized; see Keypad_Init.
Example	`kp = Keypad_Released();`

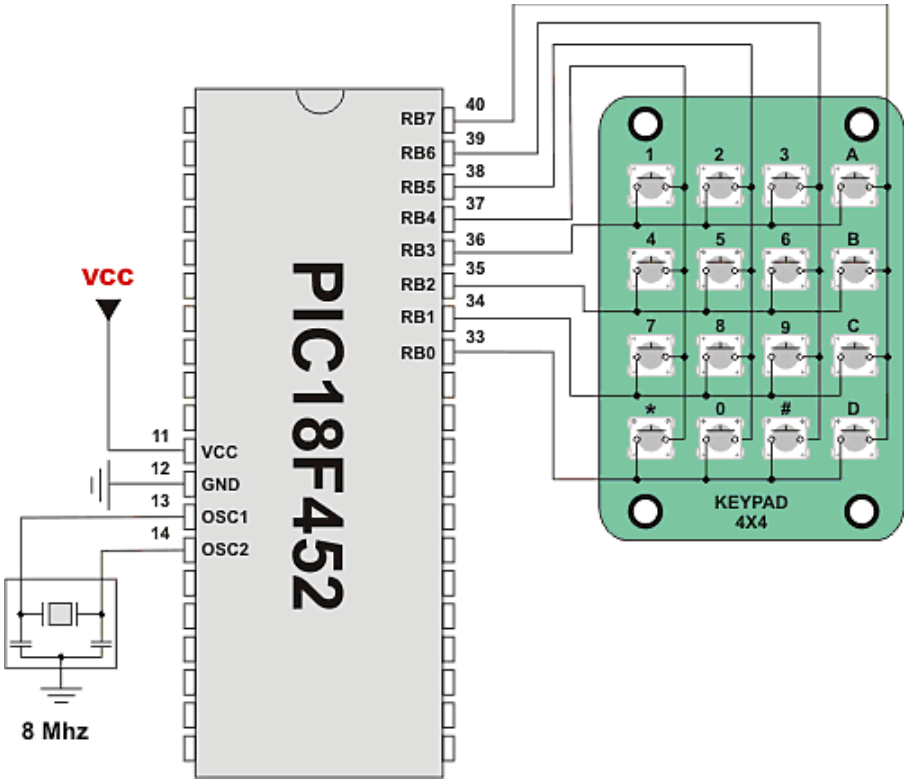
Library Example

The following code can be used for testing the keypad. It supports keypads with 1 to 4 rows and 1 to 4 columns. The code returned by the keypad functions (1..16) is transformed into ASCII codes [0..9,A..F]. In addition, a small single-byte counter displays the total number of keys pressed in the second LCD row.

```
unsigned short kp, cnt;
char txt[5];

void main() {
  cnt = 0;
  Keypad_Init(&PORTC);
  Lcd_Init(&PORTB);          // Initialize LCD on PORTC
  Lcd_Cmd(LCD_CLEAR);        // Clear display
  Lcd_Cmd(LCD_CURSOR_OFF);   // Cursor off

  Lcd_Out(1, 1, "Key  :");
  Lcd_Out(2, 1, "Times:");

  do {
    kp = 0;

    //--- Wait for key to be pressed
    do
      //--- un-comment one of the keypad reading functions
      kp = Keypad_Released();
      //kp = Keypad_Read();
    while (!kp);

    cnt++;

    //--- prepare value for output
    if (kp > 10)
```

```
      kp += 54;
    else
      kp += 47;

    //--- print it on LCD
    Lcd_Chr(1, 10, kp);
    WordToStr(cnt, txt);
    Lcd_Out(2, 10, txt);

  } while (1);
}//~!
```

HW Connection



4x4 Keypad connection scheme

ADC Library

ADC (Analog to Digital Converter) module is available with a number of PIC MCU models. Library function `Adc_Read` is included to provide you comfortable work with the module.

Adc_Read

Prototype	`unsigned Adc_Read(unsigned short channel);`
Returns	10-bit unsigned value read from the specified channel.
Description	Initializes PIC's internal ADC module to work with RC clock. Clock determines the time period necessary for performing AD conversion (min 12TAD). Parameter `channel` represents the channel from which the analog value is to be acquired. Refer to the appropriate datasheet for channel-to-pin mapping.
Requires	PIC MCU with built-in ADC module. You should consult the Datasheet documentation for specific device (most of devices from the P16 and P18 family have it). Before using the function, be sure to configure the appropriate TRISA bits to designate the pins as input. Also, configure the desired pin as analog input, and set *Vref* (voltage reference value).

Example	`unsigned tmp;`
	`...`
	`tmp = Adc_Read(1);  /* read analog value from channel 1 */`

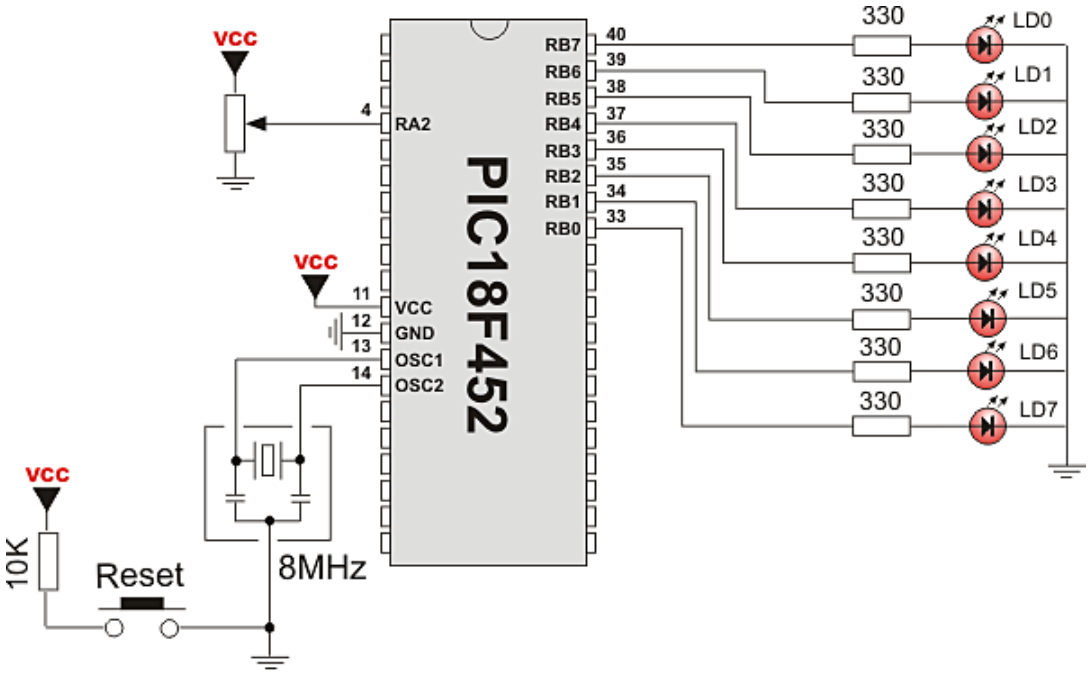
Library Example

This code snippet reads analog value from channel 2 and displays it on PORTD (lower 8 bits) and PORTB (2 most significant bits).

```
unsigned int temp_res;

void main() {
  ADCON1 = 0x80;  // Configure analog inputs and Vref
  TRISA  = 0xFF;  // PORTA is input
```

```
  TRISB  = 0x3F;  // Pins RB7, RB6 are outputs
  TRISD  = 0;     // PORTD is output

  do {
    temp_res = Adc_Read(2); // Get results of AD conversion
    PORTD = temp_res;       // Send lower 8 bits to PORTD
    PORTB = temp_res >> 2;  // Send 2 most significant bits to RB7, RB6
  } while(1);
}
```

HW Connection



ADC HW connection

PWM Library

CCP module is available with a number of PICmicros. mikroC provides library which simplifies using PWM HW Module.

Note: Certain PICmicros with two or more CCP modules, such as P18F8520, require you to specify the module you want to use. Simply append the number `1` or `2` to a Pwm. For example, `Pwm2_Start();` Also, for the sake of backward compabitility with previous compiler versions and easier code management, MCU's with multiple PWM modules have PWM library which is identical to PWM1 (i.e. you can use `PWM_Init()` instead of `PWM1_Init()` to initialize CCP1).

Library Routines

- Pwm_Init
- Pwm_Change_Duty
- Pwm_Start
- Pwm_Stop

Pwm_Init

Prototype	`void Pwm_Init(unsigned long freq);`
Returns	Nothing.
Description	Initializes the PWM module with duty ratio 0. Parameter `freq` is a desired PWM frequency in Hz (refer to device data sheet for correct values in respect with Fosc). This routine needs to be called before using other functions from PWM Library.
Requires	MCU must have CCP module.
Example	Initialize PWM module at 5KHz: `Pwm_Init(5000);`

Pwm_Change_Duty

Prototype	`void Pwm_Change_Duty(unsigned short duty_ratio);`
Returns	Nothing.
Description	Changes PWM duty ratio. Parameter `duty` takes values from 0 to 255, where 0 is 0%, 127 is 50%, and 255 is 100% duty ratio. Other specific values for duty ratio can be calculated as `(Percent*255)/100`.
Requires	MCU must have CCP module. Pwm_Init must be called before using this routine.
Example	Set duty ratio to 75%: `Pwm_Change_Duty(192);`

Pwm_Start

Prototype	`void Pwm_Start(void);`
Returns	Nothing.
Description	Starts PWM.
Requires	MCU must have CCP module. Pwm_Init must be called before using this routine.
Example	`Pwm_Start();`

Pwm_Stop

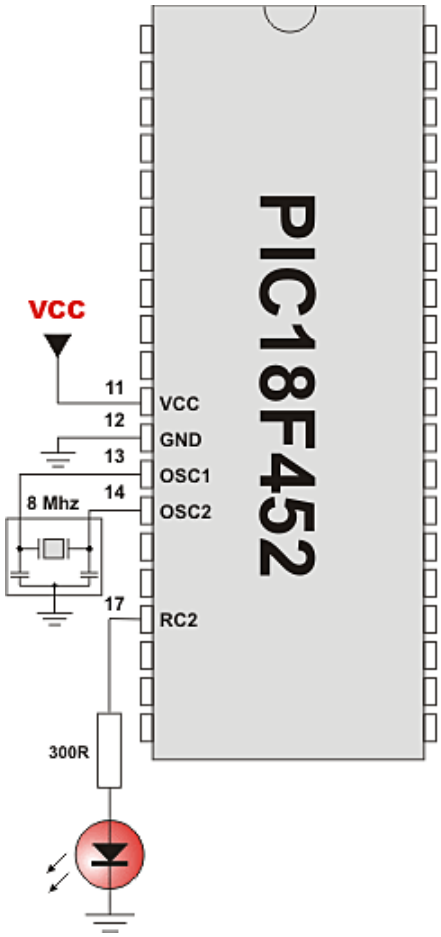
Prototype	`void Pwm_Stop(void);`
Returns	Nothing.
Description	Stops PWM.
Requires	MCU must have CCP module. Pwm_Init must be called before using this routine. Pwm_Start should be called before using this routine, otherwise it will have no effect as the PWM module is not running.
Example	`Pwm_Stop();`

Library Example

The example changes PWM duty ratio on pin RC2 continually. If LED is connected to RC2, you can observe the gradual change of emitted light.

```
// microcontroller : P16F877A
// PWM module is set on RC2.

unsigned short j, oj;

void InitMain() {
  PORTB = 0;              // Set PORTB to 0
  TRISB = 0;              // PORTB is output

  ADCON1 = 6;             // All ADC pins to digital I/O
  PORTA = 255;
  TRISA = 255;            // PORTA is input

  PORTC = 0xFF;           // Set PORTC to $FF
  TRISC = 0;              // PORTC is output
  Pwm_Init(5000);         // Initialize PWM module
}//~

void main() {
  InitMain();
  j   = 80;               // Initial value for j
  oj  = 0;                // oj will keep the 'old j' value
  Pwm_Start();            // Start PWM

  while (1) {             // Endless loop
    if (Button(&PORTA, 0,1,1))    // button on RA0 pressed
      j++ ;                       //    increment j
    if (Button(&PORTA, 1,1,1))    // button on RA1 pressed
      j-- ;                       //    decrement j

    if (oj != j) {        // If change in duty cycle requested,
      Pwm_Change_Duty(j); //   set new duty ratio,
      oj = j;             //   memorize it,
      PORTB = oj;         //   and display on PORTB
    }
    Delay_ms(200);        // Slow down a bit
  }
}
```

HW Connection



PWM demonstration

USART Library

USART hardware module is available with a number of PICmicros. mikroC USART Library provides comfortable work with the Asynchronous (full duplex) mode.

You can easily communicate with other devices via RS232 protocol (for example with PC, see the figure at the end of the topic – RS232 HW connection). You need a PIC MCU with hardware integrated USART, for example PIC16F877. Then, simply use the functions listed below.

Library Routines

- Usart_Init
- Usart_Data_Ready
- Usart_Read
- Usart_Write

Certain PICmicros with two USART modules, such as P18F8520, require you to specify the module you want to use. Simply append the number `1` or `2` to a function name. For example, `Usart_Write2();` Also, for the sake of backward compabitility with previous compiler versions and easier code management, MCU's with multiple USART modules have USART library which is identical to USART1 (i.e. you can use `Usart_Init()` instead of `Usart_Init1()` for Usart operations).

Usart_Init

Prototype	`void Usart_Init(const unsigned long baud_rate);`
Returns	Nothing.
Description	Initializes hardware USART module with the desired baud rate. Refer to the device data sheet for baud rates allowed for specific Fosc. If you specify the unsupported baud rate, compiler will report an error.
Requires	You need PIC MCU with hardware USART. `Usart_Init` needs to be called before using other functions from USART Library.
Example	This will initialize hardware USART and establish the communication at 2400 bps:

	`Usart_Init(2400);`

Usart_Data_Ready

Prototype	`unsigned short Usart_Data_Ready(void);`
Returns	Function returns 1 if data is ready or 0 if there is no data.
Description	Use the function to test if data in receive buffer is ready for reading.
Requires	USART HW module must be initialized and communication established before using this function. See Usart_Init.

Example	If data is ready, read it:
	`int receive;`
	`...`
	`if (Usart_Data_Ready()) receive = Usart_Read;`

Usart_Read

Prototype	`unsigned short Usart_Read(void);`
Returns	Returns the received byte. If byte is not received, returns 0.
Description	Function receives a byte via USART. Use the function Usart_Data_Ready to test if data is ready first.
Requires	USART HW module must be initialized and communication established before using this function. See Usart_Init.

Example	If data is ready, read it:
	`int receive;`
	`...`
	`if (Usart_Data_Ready()) receive = Usart_Read();`

Usart_Write

Prototype	`void Usart_Write(unsigned short data);`

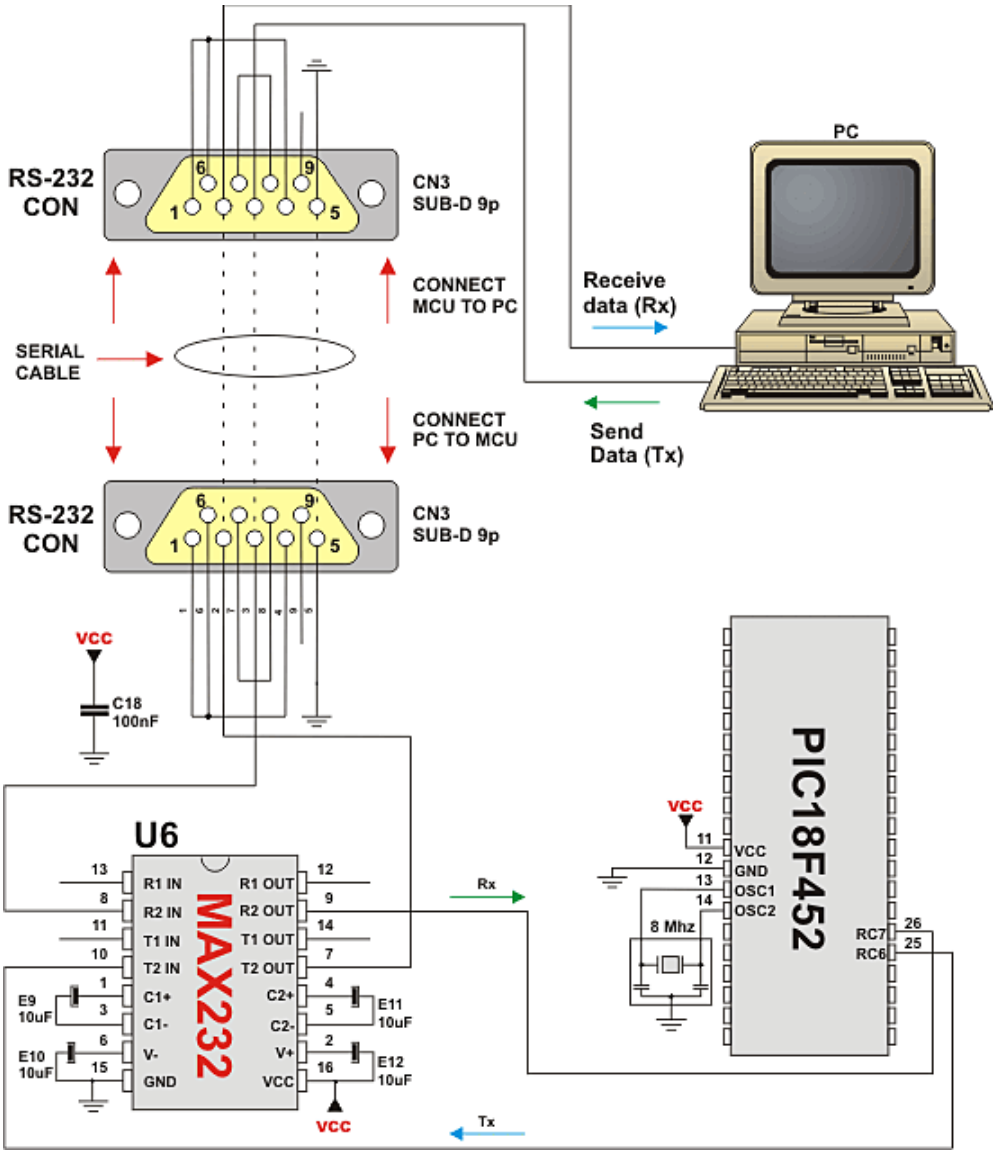
Returns	Nothing.
Description	Function transmits a byte (`data`) via USART.
Requires	USART HW module must be initialized and communication established before using this function. See Usart_Init.
Example	`int chunk = 0x1E;` `Usart_Write(chunk);  /* send chunk via USART */`

Library Example

The example demonstrates simple data exchange via USART. When PIC MCU receives data, it immediately sends the same data back. If PIC is connected to the PC (see the figure below), you can test the example from mikroC terminal for RS232 communication, menu choice **Tools › Terminal**.

```
unsigned short i;

void main() {

  // Initialize USART module (8 bit, 2400 baud rate, no parity bit..)
  Usart_Init(2400);

  do {
    if (Usart_Data_Ready()) {    // If data is received
      i = Usart_Read();          // Read the received data
      Usart_Write(i);            // Send data via USART
    }
  } while (1);
}//~!
```

HW Connection



RS232 HW connection

M. Abdelati, Fall 2007/8

Electrical & Computer Engineering Department
Microcontrollers & Microprocessors Based Systems
ECOM 4315
Midterm Exam
Duration 80 min

Name: ……..……..

Surname: ……..……...……

ID: ……..……..

Phone or email contact: ……..

Q		Grade
1	8	
2	8	
3	8	
4	8	
5	8	
4	10	
Bonus		2
Total	50	

Directions:

- Use your engineering judgment to assume/correct any missing/wrong information and don't interrupt the instructor for any reason.
- The exam is closed books and notes (instructions sheet is allowed).
- Be sure that you have a total of 6 questions.
- Attempt to solve all questions
- While extra correct information in your answers is discarded, you will lose a considerable amount of points if you provide wrong extra information.

- **Bonus Policy:** There is 2 point bonus for each student. You will lose the bonus if you
  1. ask any question of any type,
  2. barrow pencil, eraser, calculator ... etc.,
  3. set in a wrong exam room,
  4. leave the room during the exam period,
  5. show up your mobile phone,
  6. delay the submission of your answer sheets,
  7. don't answer each question in its allocated space (separate page), or
  8. ask for paper revision and don't get extra points.

- You are allowed to call the instructor once at most during the first 15 minutes. Therefore, if you want to sacrifice with the bonus and require more explanation about an exam problem then you are advised to read all exam problems and collect your questions before calling the instructor. Remember that while the instructor does not answer silly questions, you will lose the bonus for any question of any type.
- Your grade will be inversely proportional to the complexity of your approach.
- Your total grade has a saturation value of 100%.
- Any trial of cheating will trigger a disciplinary action which insures a fail in the course at least.

Good luck!

[1] Sketch a figure for a microcontroller system in which there are 2 LEDs and 2 push buttons and a 7-segment display are interfaced to a PIC16F84A microcontroller (use minimum extra components).

[2] Refer to the following Simulator:

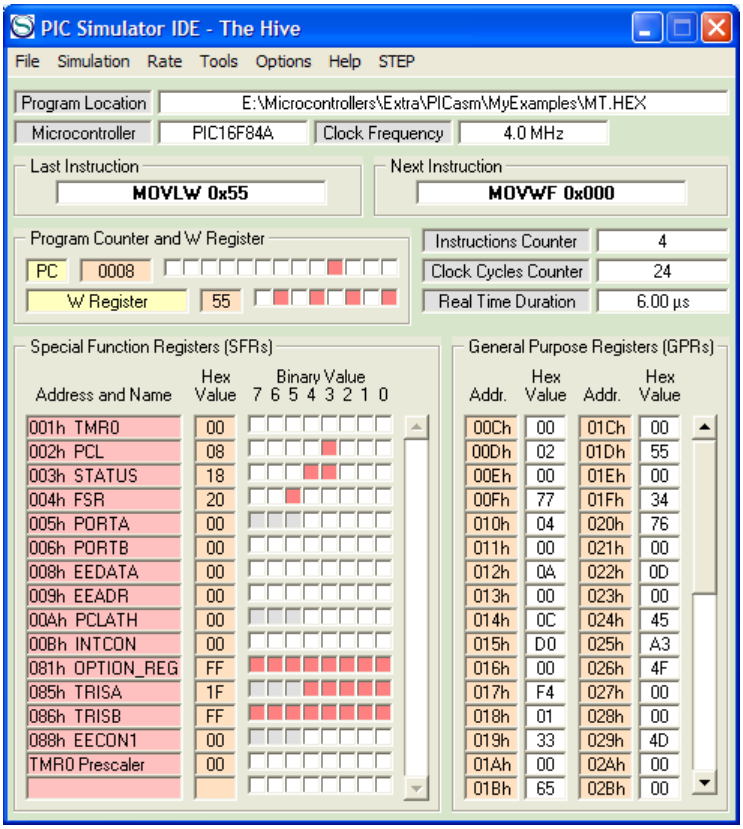
(a) What is the value of FSR register?

(b) What is the content of the general purpose register whose address is 15h

(c) What is the address of the next instruction to be executed?

(d) What is the value of the zero flag?

(e) What will happen when you execute the next instruction?



[3] Referring to PIC16F84A microcontroller, answer the following:

(a) On reset, what will be the value of the program counter (PC)?

(b) On interrupt, what will be the value of the program counter (PC)?

(c) What is the width of the instruction bus?

(d) What is the maximum program size?

(e) What is the difference between an instruction and a directive in assembly language?

(f) What are the registers involved during writing to the EEPROM?

(g) Why the register TMR0 is also called Counter register?

(h) What is the most probable execution time of PIC16F84A instructions. (given that its oscillator frequency is 4 MHz)

[4] convert the folowing assembly code to machine language and fill the resultant code in the memory model shown below.

```
          ORG  10h
LOOP      MOVF  PORTA,W
          MOVWF PORTB
          GOTO  LOOP
          END
```

address	content
0000	
0001	
0002	
0003	
0004	
0005	
0006	
0007	
0008	
0009	
000A	
000B	
000C	
000D	
000E	
000F	
0010	
0011	
0012	
0013	
0014	
0015	
0016	
0017	
0018	
0019	
001A	

[5] The following program is described at class under the name "Ramadan2".

(a) What is the function of this program?

(b) If the constant in the instruction pointed by the arrow is replaced by (.254), then the program will not function properly. However, if it is replaced by (.253) the program functions as required!

What is the cause of this bug? Describe a method to resolve this bug?

```
        LIST   P=PIC16F84A
        #INCLUDE <P16F84A.INC>
        __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

        ORG    0X000
        GOTO   START

        ORG    0X004
        RETFIE

START   BSF    STATUS,RP0          ;BANK1
        BCF    TRISB,0             ;RB0 IS OUTPUT
        BCF    STATUS,RP0          ;BANK0
        MOVLW  01h
LOOP    XORWF  PORTB, F
        CALL   DELAY
        GOTO   LOOP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
DELAY   MOVLW  .255
        MOVWF  0CH
LOOP1   NOP
        MOVLW  .255 <--
        MOVWF  0DH
LOOP0   NOP
        DECFSZ 0DH
        GOTO   LOOP0
        DECFSZ 0CH
        GOTO   LOOP1
        RETURN
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        END
```

[6] Write assembly language **instructions** to clear the general purpose registers of the PIC16F84A microcontroller (don't write the whole program).

M. Abdelati, Fall 2007/8

Electrical & Computer Engineering Department
Microcontrollers & Microprocessors Based Systems
ECOM 4315
Final Exam
Duration 120 min

Q		Grade
1	10	
2	10	
3	10	
4	20	
5		
Bonus		2
Total	50	

Name: ……..……..

Surname: ……..……...……

ID: ……..……..

Phone or email contact: ……..

Directions:

- Use your engineering judgment to assume/correct any missing/wrong information and don't interrupt the instructor for any reason.
- The exam is closed books and notes (instructions sheet is allowed).
- Be sure that you have a total of 5 questions.
- Attempt to solve all questions
- While extra correct information in your answers is discarded, you will lose a considerable amount of points if you provide wrong extra information.

- **Bonus Policy:** There is 2 point bonus for each student. You will lose the bonus if you
  1. ask any question of any type,
  2. barrow pencil, eraser, calculator ... etc.,
  3. set in a wrong exam room,
  4. leave the room during the exam period,
  5. show up your mobile phone,
  6. delay the submission of your answer sheets,
  7. don't answer each question in its allocated space (separate page), or
  8. ask for paper revision and don't get extra points.

- You are allowed to call the instructor once at most during the first 15 minutes. Therefore, if you want to sacrifice with the bonus and require more explanation about an exam problem then you are advised to read all exam problems and collect your questions before calling the instructor. Remember that while the instructor does not answer silly questions, you will lose the bonus for any question of any type.
- Your grade will be inversely proportional to the complexity of your approach.
- Your total grade has a saturation value of 100%.
- Any trial of cheating will trigger a disciplinary action which insures a fail in the course at least.

Good luck!

[1]

a) Sketch the pin diagram of the PIC16F84A

b) Sketch a figure which illustrates the internal architecture of the PIC16F84A. Don't forget to indicate the size of busses.

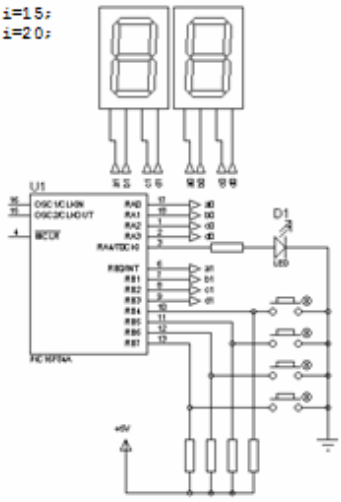
c) Write down 5 tools we have used during the course and indicate the function of each tool.

d) Write an assembly language program to add the constant h'AA' to the contents of File h'36' and store the result in File h'40'. Remember that File means *data memory location*.

[2] Assuming that you have found a 1-second delay subroutine (OneSecond) for the PIC16F84 microcontroller, write a macro that would give a 1-minute delay.

[3] At class we discussed the program shown below. Modify the program so that the LED will flash for 1 minute after the termination of timer countdown.

```
//Iqama Timers
int i, j;

void interrupt()
{
  INTCON=INTCON & 0b11111110;
  if ((portb & 0b00010000)==0b0) i=10;
  if ((portb & 0b00100000)==0b0) i=15;
  if ((portb & 0b01000000)==0b0) i=20;
}

void main()
{
  INTCON   = 0b10001000;
  PORTA    = 0;
  TRISA    = 0;
  PORTB    = 0;
  TRISB    = 0xF0;
  while(1)
   {
    while(i>0)
        {
           j = i % 10;
           PORTA = j;
           j = (i / 10) % 10;
           PORTB = j;
           Delay_us(995840);
           i-- ;
        }
    porta=0;
    portb=0;
   }
}
```



[4] In an open-loop speed control of a DC motor application it is desired to specify the set point using 8 dip switches so that the specified speed will cover the range [0-255]. Assume that the controller will be based on PIC16F84A and will utilize PWM with period equals 255 micro second.

a) Sketch the system hardware.
b) Sketch the PWM signal when the set speed = 150 .
c) Write the necessary software.

[5]

a) Have you attended the department workshop last month.?

b) If you answered yes in part a, have you submitted a report?

c) If you answered yes in part b, write down 3 comments about the workshop.

d) Have you submitted a report summarizing the course material?

e) Who is your partner (if any)? What part of that assignment you have done by yourself and what part is done by your partner?

f) Have you submitted a CD?

g) If yes, describe how you organized it.

This question is for computer engineering students. It is considered as part of question 1 and hence its grade will be included there.

Below is a segment of code written in PIC16F84 assembly language:

```
                movlw d'20'
                movwf h'3F'
LOOP1           movlw d'250'
LOOP2           addlw -1
                btfss STATUS,Z
                goto LOOP2
                decf h'3F',f
                btfss STATUS,Z
                goto LOOP1
                sleep
```

(i) Explain what this routine does and suggest its most likely purpose.

(ii) Determine the execution time of the complete program segment, assuming a clock rate of 4 MHz.

Prof. M. Abdelati, Fall 2008/9

Electrical & Computer Engineering Department
Microcontrollers & Microprocessors Based Systems
ECOM 4315
Midterm Exam
Duration 80 min

Name: ……..……..

Surname: …….……...……

ID: …….……..

Phone or email contact: ……..

Q		Grade
1	10	
2	30	
Bonus		2
Total	40	

Directions:

- Use your engineering judgment to assume/correct any missing/wrong information and don't interrupt the instructor for any reason.
- The exam is closed books and notes (instructions sheet is allowed).
- Be sure that you have a total of 6 questions.
- Attempt to solve all questions
- While extra correct information in your answers is discarded, you will lose a considerable amount of points if you provide wrong extra information.

- **Bonus Policy:** There is 2 point bonus for each student. You will lose the bonus if you
    1. ask any question of any type,
    2. barrow pencil, eraser, calculator ... etc.,
    3. set in a wrong exam room,
    4. leave the room during the exam period,
    5. show up your mobile phone,
    6. delay the submission of your answer sheets,
    7. don't answer each question in its allocated space (separate page), or
    8. ask for paper revision and don't get extra points.

- You are allowed to call the instructor once at most during the first 15 minutes. Therefore, if you want to sacrifice with the bonus and require more explanation about an exam problem then you are advised to read all exam problems and collect your questions before calling the instructor. Remember that while the instructor does not answer silly questions, you will lose the bonus for any question of any type.
- Your grade will be inversely proportional to the complexity of your approach.
- Your total grade has a saturation value of 100%.
- Any trial of cheating will trigger a disciplinary action which insures a fail in the course at least.

Good luck!

[1] Sketch a figure for a simple circuit used to test a PIC16F84A microcontroller. Write the associated software program.

[2] Referring to PIC16F84A, answer the following:

2.1. At what addresses are the Reset Vector and Interrupt Vector?

2.2.How many words of program memory are available in the PIC?

2.3.Which registers set the directions of the digital I/O pins?

2.4.How many registers are hard-wired across the two banks?

2.5.What does the following opcode do? 00 0000 0110 0000

2.6.How many different prescale assignments are available for Timer0?

2.7.Where are the bank select bits?

2.8.What is W and STATUS/Z after the following instruction sequence?

```
MOVLW 1
SUBLW 1
```

2.9.What is the difference between a microprocessor and a microcontroller?

2.10. How many different interrupt sources does the PIC have?

2.11. What is the difference between machine code and assembly code?

2.12. Write a two-instruction sequence to take the two's complement of PortB.

2.13. The following code has an infinite loop. Why?

```
        MOVLW   3
        MOVWF   PORTB
LOOP:   DECFSZ  PORTB, W
        GOTO    LOOP
```

2.14. What does RISC stand for?

2.15. The following code was written to copy PORTB to TRISA. Find the bug.

```
MOVF    PORTB, W
MOVWF   TRISA
```

2.16. The following routine toggles RB2 after an interrupt on RB0/INT. Find the bug.

```
ORG     0X004
MOVWF   0Eh
SWAPF   STATUS,W
MOVWF   0Fh

BCF     STATUS,RP0
MOVLW   4
XORWF   PORTB,F
BCF     INTCON,INTF

SWAPF   0Fh,W
MOVWF   STATUS
SWAPF   0EH,F
SWAPF   0EH,W
RETURN
```

2.17. What is the Harvard architecture?

2.18. Suppose STATUS = 0x10. Which bank is selected?

2.19. Where is the configuration word stored?

2.20. What is the machine code for the instruction CLR PORTA, F? Give your answer in hex. What register will this clear if STATUS = 0011 1000?

Prof. M. Abdelati & Mr. M. Almobaied Fall 2008/9

Electrical Engineering Department
Microcontrollers & Microprocessors Based Systems
ECOM 4315
Final Exam
Duration 120 min

Q		Grade
1	7	
2	7	
3	6	
4	10	
5	10	
6	10	
Bonus		2
Total	50	

Name: ……..……..

Surname: ……..……...……

ID: ……..……..

Phone or email contact: ……..

Directions:

- Use your engineering judgment to assume/correct any missing/wrong information and don't interrupt the instructor for any reason.
- The exam is closed books and notes (instructions sheet is allowed).
- Be sure that you have a total of 6 questions.
- Attempt to solve all questions
- While extra correct information in your answers is discarded, you will lose a considerable amount of points if you provide wrong extra information.

- **Bonus Policy:** There is 2 point bonus for each student. You will lose the bonus if you
  1. ask any question of any type,
  2. barrow pencil, eraser, calculator ... etc.,
  3. set in a wrong exam room,
  4. leave the room during the exam period,
  5. show up your mobile phone,
  6. delay the submission of your answer sheets,
  7. don't answer each question in its allocated space (separate page), or
  8. ask for paper revision and don't get extra points.

- You are allowed to call the instructor once at most during the first 15 minutes. Therefore, if you want to sacrifice with the bonus and require more explanation about an exam problem then you are advised to read all exam problems and collect your questions before calling the instructor. Remember that while the instructor does not answer silly questions, you will lose the bonus for any question of any type.
- Your grade will be inversely proportional to the complexity of your approach.
- Your total grade has a saturation value of 100%.
- Any trial of cheating will trigger a disciplinary action which insures a fail in the course at least.

Good luck!

[1] The following questions are True-False questions referred to PIC16F84A. If the statement is true, circle T, and go to the next question. If the statement is false, circle F, write the specific portion of the statement that makes it false in space A, and rewrite the incorrect portion so that it is true in space B.

T F 1. The instruction bus consists of 14 bit .

A_____

B_____

T F 2. EECON2 is an existing register with 8 bits used in writing in the EEPROM.

A_____

B_____

T F 3. Disabling the GIE the pc will never reach the interrupt subroutine.

A_____

B_____

T F 4. To read from the EEPROM we need to use (EEADR , EEDATA , EECON1, EECON2).

A_____

B_____

T F 5. we can write any code in assembly language in the microC without any restrictions.

A_____

B_____

T F 6. The addresses of EEPROM memory starts at location 0x00h & extend to 0x3f but consists of 64 byte.

A_____

B_____

T F 7. Using Macros are preferable than using subroutines.

A_____

B_____

[2] Referring to PIC16F84A, choose the most suitable answer:

1. The external recourses of the interrupts are:
   a. RB0 & RB4-7
   b. TIMER ZERO
   c. Finishing writing on the EEPROM.
   d. All the above is true.

2. To disable all the interrupts we can
   a. put 1 in the GIE .
   b. put 0 in the GIE .
   c. disable EEIE , T0IE , RBIE & INTIE
   d. a & c

3. To select the prescaler for the WDT we assign (PSA) to be :
   a. 1
   b. 0
   c. 0xF h
   d. Don't care.

4. The instruction **`decfsz`** is ……. cycle instruction.
   a. one
   b. two
   c. one or two depending on the result
   d. one or two depending on the initial value .

5. The indirect addressing is accomplished by using :
   a. FSR register
   b. INDF register
   c. Both a &b
   d. None of the above.

6. Machine code is:
   a. The Instruction decoder in the processor that interprets the operation code of the instruction.
   b. Assembly-level symbolic source code.
   c. The binary address of the cell in program memory an instruction lies.
   d. The binary code in Program memory that the processor executes.

7. A subroutine is:
   a. A program that links software routines into one executable program.
   b. A program that allows you to enter the source program as a file on disc.
   c. A program module that performs a well defined function.
   d. A program that translates from assembly-level source to machine code.

[3] Write a PIC assembly-level code to compare the two numbers NUMBER1 in File h'30' and NUMBER2 in File h'31' and if these are equal add 6 onto DATUM in File h'40', otherwise decrement it.

[4] Write a macro which fills the even bytes and the odd bytes of the General Purpose registers with the numbers 0x55h and 0xAAh respectively.

[5] DC servo motors are characterized by simplicity in speed and position control applications. They are classified into two categories depending on the feedback signal returned from the motor. One type has a tachometer which returns back a voltage signal depending on the rotation speed while the other type has an encoder which returns back pulses depending on the rotation angle. Show how you can interface each type (separately) to PIC16F84A. Show the hardware only and don't write any software.

[6] A simple alarm system has 4 zone inputs connected to PIC16F84A. If any one of these inputs is activated a bell will sound for 5 minutes and the zone number will be displayed on a 7-segment display.

a. Provide a hardware design for the system.
b. Write a C code to implement the required functionality of the system.

Prof. M. Abdelati, Summer 2008/9

Electrical Engineering Department
Microcontrollers & Microprocessors Based Systems
ECOM 4315
Midterm Exam
Duration 90 min

Name: ……..……..

Surname: ……..……

ID: ……..

Phone or email contact: ……..

Q		Grade
1	10	
2	10	
3	10	
4	10	
Bonus		2
Total	30	

Directions:

- Use your engineering judgment to assume/correct any missing/wrong information and don't interrupt the instructor for any reason.
- The exam is closed books and notes (instructions sheet is allowed).
- Be sure that you have a total of 4 questions.
- Attempt to solve all questions
- While extra correct information in your answers is discarded, you will lose a considerable amount of points if you provide wrong extra information.

- **Bonus Policy:** There is 2 point bonus for each student. You will lose the bonus if you
  1. ask any question of any type,
  2. barrow pencil, eraser, calculator ... etc.,
  3. set in a wrong exam room,
  4. leave the room during the exam period,
  5. show up your mobile phone,
  6. delay the submission of your answer sheets,
  7. don't answer each question in its allocated space (separate page), or
  8. ask for paper revision and don't get extra points.

- You are allowed to call the instructor once at most during the first 15 minutes. Therefore, if you want to sacrifice with the bonus and require more explanation about an exam problem then you are advised to read all exam problems and collect your questions before calling the instructor. Remember that while the instructor does not answer silly questions, you will lose the bonus for any question of any type.
- Your grade will be inversely proportional to the complexity of your approach.
- Your total grade has a saturation value of 100%.
- Any trial of cheating will trigger a disciplinary action which insures a fail in the course at least.

Good luck!

[1] Sketch the circuit diagram of a microcontroller system in which there are 2 LEDs and 2 push buttons and two 7-segment displays are interfaced to a PIC16F84A microcontroller.

[2] The following record exists is a hex file

:0800080009008316B8308100E5

Extract the assembly language instructions from the record given that it has the following format

Record Mark ':'	Load Record Length	Offset	Record Type	Information or Data	Chick Sum
1-byte	1-byte	2-bytes	1-byte	n-bytes	1-byte

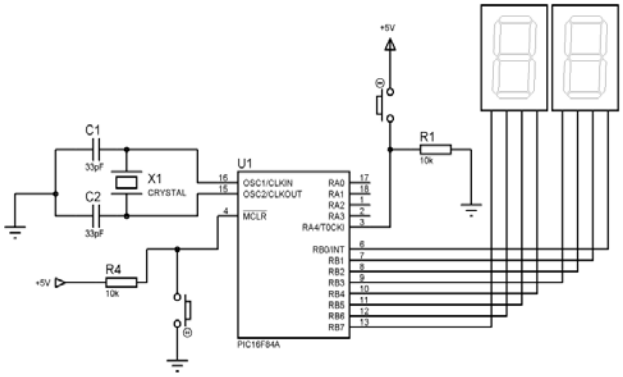
Machine Code (hexa)	Machine Code (binary)	Assembly code

Prof. M. Abdelati, Summer 2008/9

[3] Write two macros called EEread and EEwrite that allows reading and writing from the EEPROM. The EEread macro takes the address as an argument and returns the data in the work register. The EEwrite macro takes the address and data as arguments.

```
; Ex7 (EE_a) EEPROM handling applica
            LIST   P=PIC16F84A
            #INCLUDE <P16F84A.INC>
            __CONFIG  _XT_OSC&_PWRTE

            ORG    0X000
            GOTO   START
            ORG    0X004
            RETFIE

START       BSF    STATUS,RP0
            CLRF   TRISB
            BSF    TRISB,0
            MOVLW  B'00011111'
            MOVWF  TRISA

            BCF    STATUS,RP0
            MOVLW  07h
            MOvWF  EEADR

LOOP0       BTFSS  PORTB,0
            GOTO   LOOP0
            MOVF   PORTA,W
            MOVWF  EEDATA

            BSF    STATUS,RP0
            BCF    INTCON,GIE
            BCF    EECON1,EEIF
            BSF    EECON1,WREN
            MOVLW  55h
            MOVWF  EECON2
            MOVLW  0AAh
            MOVWF  EECON2
            BSF    EECON1,WR
            BSF    INTCON,GIE

LOOP1       BTFSS  EECON1,EEIF
            GOTO   LOOP1

            BCF    STATUS,RP0
            CLRF   EEDATA

LOOP2       BTFSC  PORTB,0
            GOTO   LOOP2

            BSF    STATUS,RP0
            BSF    EECON1,RD
            BCF    TRISB,0
            BCF    STATUS,RP0
            MOVF   EEDATA,W
            MOVWF  PORTB

LOOP3       GOTO   LOOP3
            END
```

[4]




```
; Ex5 (Counter a) Counter on RA4/T0CKI from 00-FF
; No Interrupt is used here

                LIST    P=PIC16F84A
                #INCLUDE <P16F84A.INC>
                __CONFIG  _XT_OSC&_PWRTE_ON&_CP_OFF&_WDT_OFF

                ORG     0X000
                GOTO    START
                ORG     0X004
                RETFIE

START           BSF     STATUS,RP0              ; BANK 1
                MOVLW   B'10111000'
                MOVWF   OPTION_REG      ; xx11 1xxx
                CLRF    TRISB           ; PORT B IS OUTPUT
                MOVLW   B'00011111'
                MOVWF   TRISA           ; PORT A IS INPUT
                BCF     STATUS,RP0      ; BANK0

                CLRF    TMR0
LOOP            MOVF    TMR0,W
                MOVWF   PORTB
                GOTO    LOOP
                END
```

Modify the hardware and software of the example shown above so that the same task is done using interrupt mechanism.

Prof. M. Abdelati, Summer 2008/9

Electrical Engineering Department
Microcontrollers & Microprocessors Based Systems
ECOM 4315

Final Exam
Duration 120 min

Q		Grade
1	10	
2	10	
3	10	
4	10	
5	10	
Bonus		2
Total	50	

Name: ……..……..

Department: ……..……

ID: ……..

Phone or email contact: ……..

Directions:

- Use your engineering judgment to assume/correct any missing/wrong information and don't interrupt the instructor for any reason.
- The exam is closed books and notes (instructions sheet is allowed).
- Be sure that you have a total of 5 questions.
- Attempt to solve all questions
- While extra correct information in your answers is discarded, you will lose a considerable amount of points if you provide wrong extra information.

- **Bonus Policy:** There is 2 point bonus for each student. You will lose the bonus if you
  1. ask any question of any type,
  2. barrow pencil, eraser, calculator ... etc.,
  3. set in a wrong exam room,
  4. leave the room during the exam period,
  5. show up your mobile phone,
  6. delay the submission of your answer sheets,
  7. don't answer each question in its allocated space (separate page), or
  8. ask for paper revision and don't get extra points.

- You are allowed to call the instructor once at most during the first 15 minutes. Therefore, if you want to sacrifice with the bonus and require more explanation about an exam problem then you are advised to read all exam problems and collect your questions before calling the instructor. Remember that while the instructor does not answer silly questions, you will lose the bonus for any question of any type.
- Your grade will be inversely proportional to the complexity of your approach.
- Your total grade has a saturation value of 100%.
- Any trial of cheating will trigger a disciplinary action which insures a fail in the course at least.

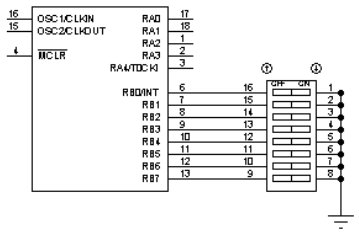
Good luck!

[1] Sketch the circuit diagram of a PIC16F877A microcontroller system in which the microcontroller is interfaced to the following peripherals:

sn	Item
1	Hexadecimal keypad.
2	Bidirectional DC motor 48 V
3	LED
4	Toggle switch
5	LCD
6	4 Multiplexed Seven Segment Displays
7	Temperature sensor

[2] Write a subroutine which takes an 8-bit number in the work register. The input number has only one single non-zero bit. The subroutine should return a number from 0 to 7 indicating the position of the non zero bit. For example, if the input is 00010000 then the output is 4. The output is returned in the work register.

[3] a) A student has connected a dipswitch to RB like this: What are your comments?



b) Assume that you have a Mico C project files folder

i) What are the extensions of the essential files?

ii) If you want to rename the project, can you rename those essential files only? If no, what must be done?

[4] What are the course headlines? And what are the contributions of the students?

[5] (For EE students only) You have 2 push buttons connected to A0 and A1 of a 16F84A microcontroller and they will be used to enter a number from 0 to 100 into a variable x. The input A1 is used for decreasing while A0 is used for increasing the input number. Show the Mico C code.

[5] (For CE students only) Propose a method to test the serial port of PIC16F877 if your PC doesn't have a serial port. Show the circuit diagram and write the required program.

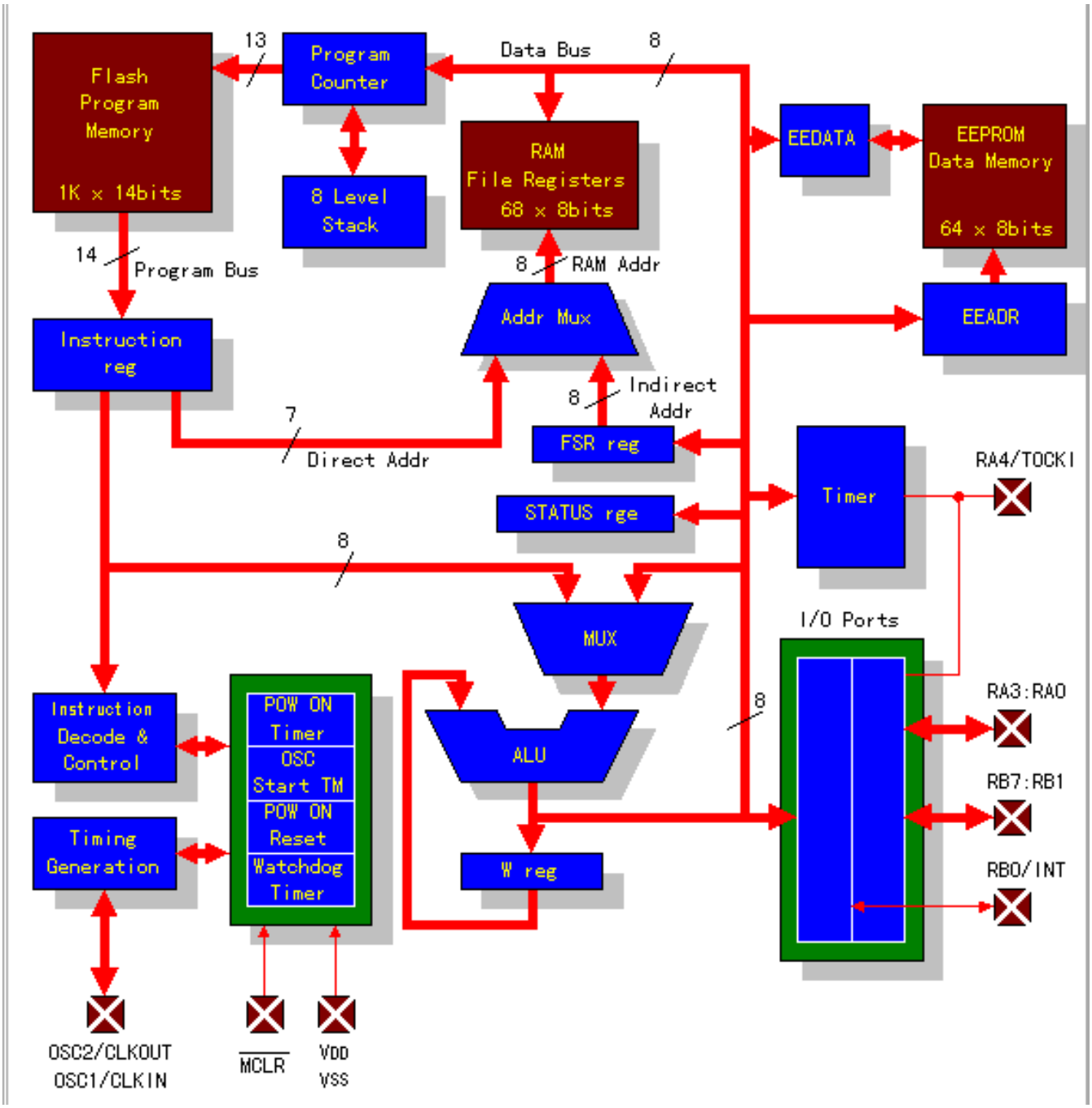

Flash
Program
Memory
1K x 14bits
13
Program
Counter
Data Bus
8
8 Level
Stack
RAM
File Registers
68 x 8bits
EEDATA
EEPROM
Data Memory
64 x 8bits
14
Program Bus
8
RAM Addr
Addr Mux
EEADR
Instruction
reg
8
Indirect
Addr
7
Direct Addr
FSR reg
STATUS rge
Timer
RA4/TOCKI
8
MUX
I/O Ports
Instruction
Decode &
Control
POW ON
Timer
OSC
Start TM
POW ON
Reset
Watchdog
Timer
ALU
8
RA3:RA0
RB7:RB1
RB0/INT
Timing
Generation
W reg
OSC2/CLKOUT
OSC1/CLKIN
MCLR
VDD
VSS

TABLE 2-1: SPECIAL FUNCTION REGISTER FILE SUMMARY

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on RESET	Details on page
Bank 0											
00h	INDF	Uses contents of FSR to address Data Memory (not a physical register)								---- ----	11
01h	TMR0	8-bit Real-Time Clock/Counter								xxxx xxxx	20
02h	PCL	Low Order 8 bits of the Program Counter (PC)								0000 0000	11
03h	STATUS(2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	8
04h	FSR	Indirect Data Memory Address Pointer 0								xxxx xxxx	11
05h	PORTA(4)	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	16
06h	PORTB(5)	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	18
07h	—	Unimplemented location, read as '0'								—	—
08h	EEDATA	EEPROM Data Register								xxxx xxxx	13,14
09h	EEADR	EEPROM Address Register								xxxx xxxx	13,14
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of the PC(1)					---0 0000	11
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10
Bank 1											
80h	INDF	Uses Contents of FSR to address Data Memory (not a physical register)								---- ----	11
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	9
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	11
83h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	8
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	11
85h	TRISA	—	—	—	PORTA Data Direction Register					---1 1111	16
86h	TRISB	PORTB Data Direction Register								1111 1111	18
87h	—	Unimplemented location, read as '0'								—	—
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	13
89h	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	14
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC(1)					---0 0000	11
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0', q = value depends on condition

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> are never transferred to PCLATH.

2: The $\overline{TO}$ and $\overline{PD}$ status bits in the STATUS register are not affected by a $\overline{MCLR}$ Reset.

3: Other (non power-up) RESETS include: external RESET through $\overline{MCLR}$ and the Watchdog Timer Reset.

4: On any device RESET, these pins are configured as inputs.

5: This is the value that will be in the port output latch.

A-2 PIC 16Fxx Instruction Set

Mnemonic, Operands		Description	Cyc.	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
DDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$, $\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}$, $\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Note：
1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.
2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.
3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

Table A-2
PIC 16Fxx Instruction Set.